Revision 1.0
Last Print Date: 10/2/2002 12:00 PM

Intel® PRO Network Adapters WMI Provider User's Guide

Legal Notices and Disclaimers

Information in this document is provided in connection with Intel® products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL® PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, or life sustaining applications. Intel may make changes to specifications and product descriptions at any time, without notice.

This manual as well as the software described in it is furnished under license and may only be used or copied in accordance with the terms of the license. The information in this manual is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Intel Corporation. Intel Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this document or any software that may be provided in association with this document.

WARNING: This product contains information that could be used to attack and/or disable a computer system(s) or network(s). A thorough knowledge of Microsoft operating system security features should be a prerequisite to any implementation of this product, and developers and users are strongly encouraged to contact Microsoft regarding any security concerns that they may have prior to using any implementation of this product in a production environment.

Revision History

Rev.	Date	Description
1.0	10/2/2002	Initial release.

1. Introduction 8
1.1 Scope 8
1.2 Related Documents 8
2. WMI 9
2.1 Common Information Model (CIM Schema) 10
3. Main Features 11
3.1 Implemented Features 11
3.1.1 Adapter 11
3.1.2 Team 11
3.1.3 VLAN 11
3.1.4 Event Notification 11
4. Installed Files 12
4.1 Executables 12
4.2 MOF files 12
4.2.1 MOF files for IntelNCS namespace: 12
4.2.2 MOF files for CIMV2 namespace: 12
4.3 Resource Files 12
5. Security 13
6. Namespace and Context 14
6.1 Namespace 14
6.2 WBEM Context 14
7. Locales and Localization 16
7.1 Localized MOF files 16
7.1.1 Class Storage 16
7.1.2 Localized MOF support in cimv2 namespace 16
7.1.3 Runtime support 16
7.2 Localized attribute data 16
8. Error Reporting 17
8.1 IANet_ExtendedStatus 17
8.1.1 Getting the error object 17
8.1.2 Error Object Qualifiers 17
8.2 Error Codes 17
9. The Core Schema 20
9.1 Schema Diagram 20
9.2 IANet_NetService 20
9.2.1 Purpose 20
9.2.2 Instances 20
9.2.3 Creating instances 20
9.2.4 Removing instances 20
9.2.5 Modifying properties 21
9.2.6 Supported Attributes 21
9.2.7 Methods 21
9.3 Core Events 21
9.3.1 IANet_SessionEvent 21
9.3.2 Purpose 21
9.3.3 Triggers 21
9.3.4 Event Data 21
9.3.5 Purpose 22
9.3.6 Triggers 22
9.3.7 Event Data 22
9.4 Use Cases 22

9.4.1 Getting a Session Handle 22
9.4.2 Using a session handle in the IWbemContext object 23
9.4.3 Reading pending changes using a session handle 23
9.4.4 Finishing with a session handle 23
9.4.5 Registering for the Core Events 23
10. Ethernet Adapter Schema 24
10.1 Schema Diagram 24
10.2 IANet_EthernetAdapter 24
10.2.1 Purpose 24
10.2.2 Instances 24
10.2.3 Creating instances 25
10.2.4 Removing instances 25
10.2.5 Modifying properties 25
10.2.6 Unsupported Attributes 25
10.2.7 Methods: 25
10.3 IANet_IPProtocolEndpoint 25
10.3.1 Purpose 26
10.3.2 Instances 26
10.3.3 Creating instances 26
10.3.4 Removing instances 26
10.3.5 Modifying properties 26
10.3.6 Associations 26
10.3.7 Supported Attributes 26
10.3.8 Unsupported Attributes 26
10.3.9 Methods: 27
10.4 IANet_BootAgent 27
10.4.1 Purpose 27
10.4.2 Instances 27
10.4.3 Creating instances 27
10.4.4 Removing instances 27
10.4.5 Modifying properties 27
10.4.6 Associations 27
10.4.7 Supported Attributes 27
10.4.8 Unsupported Attributes 27
10.4.9 Methods 27
10.5 IANet_PCIDevice 28
10.5.1 Purpose 28
10.5.2 Instances 28
10.5.3 Creating instances 28
10.5.4 Removing instances 28
10.5.5 Modifying properties 28
10.5.6 Associations 28
10.5.7 Methods 28
10.5.8 Unsupported Attributes 28
10.6 IANet_EthernetPCIDevice 28
10.6.1 Purpose 28
10.6.2 Instances 28
10.6.3 Creating instances 29
10.6.4 Removing instances 29
10.6.5 Modifying properties 29
10.6.6 Associations 29
10.6.7 Unsupported Attributes 29
10.6.8 Methods 29
11. Setting Schema 30
11.1 Schema Diagram 30
11.2 IANet_Configuration 30

11.2.1 Purpose 30
11.2.2 Instances 30
11.2.3 Creating instances 30
11.2.4 Removing instances 31
11.2.5 Modifying properties 31
11.2.6 Associations 31
11.2.7 Methods 31
11.2.8 Unsupported Attributes 31
11.3 IANet_Setting 31
11.3.1 Purpose 31
11.3.2 Instances 31
11.3.3 Creating instances 31
11.3.4 Removing instances 31
11.3.5 Modifying properties 31
11.3.6 Associations 31
11.3.7 Methods 32
11.3.8 Unsupported Attributes 32
11.4 IANet_SettingInt 32
11.4.1 Purpose 32
11.4.2 Instances 32
11.4.3 Creating instances 32
11.4.4 Removing instances 32
11.4.5 Modifying properties 32
11.4.6 Associations 32
11.4.7 Unsupported Attributes 32
11.4.8 Methods 32
11.5 IANet_SettingEnum 33
11.5.1 Purpose 33
11.5.2 Instances 33
11.5.3 Creating instances 33
11.5.4 Removing instances 33
11.5.5 Modifying properties 33
11.5.6 Associations 33
11.5.7 Unsupported Attributes 33
11.5.8 Methods 33
11.6 IANet_SettingSlider 33
11.6.1 Purpose 33
11.6.2 Instances 33
11.6.3 Creating instances 33
11.6.4 Removing instances 34
11.6.5 Modifying properties 34
11.6.6 Associations 34
11.6.7 Unsupported Attributes 34
11.6.8 Methods 34
11.7 IANet_SettingMultiSelection 34
11.7.1 Purpose 34
11.7.2 Instances 34
11.7.3 Creating instances 34
11.7.4 Removing instances 34
11.7.5 Modifying properties 34
11.7.6 Associations 34
11.7.7 Unsupported Attributes 35
11.7.8 Methods 35
11.8 IANet_SettingString 35
11.8.1 Purpose 35
11.8.2 Instances 35

11.8.3 Creating instances ... 35
11.8.4 Removing instances ... 35
11.8.5 Modifying properties ... 35
11.8.6 Associations ... 35
11.8.7 Methods ... 35
11.8.8 Unsupported Attributes ... 35
11.8.9 Methods ... 35
12. Team Schema ... 36
12.1 Schema Diagram ... 36
12.2 IANet_TeamOfAdapters ... 36
12.2.1 Purpose ... 36
12.2.2 Instances ... 36
12.2.3 Creating instances ... 36
12.2.4 Removing instances ... 37
12.2.5 Modifying properties ... 37
12.2.6 Associations ... 37
12.2.7 Methods ... 37
12.3 IANet_TeamedMemberAdapter ... 37
12.3.1 Purpose ... 37
12.3.2 Instances ... 37
12.3.3 Creating instances ... 37
12.3.4 Removing instances ... 37
12.3.5 Modifying properties ... 37
12.3.6 Associations ... 38
12.3.7 Methods ... 38
12.4 IANet_NetworkVirtualAdapter ... 38
12.4.1 Purpose ... 38
12.4.2 Instances ... 38
12.4.3 Creating instances ... 38
12.4.4 Removing instances ... 38
12.4.5 Associations ... 38
12.4.6 Methods ... 38
13. VLAN Schema ... 39
13.1 Schema Diagram ... 39
13.2 IANet_802dot1QVLANService ... 39
13.2.1 Purpose ... 39
13.2.2 Instances ... 39
13.2.3 Creating instances ... 40
13.2.4 Removing Instances ... 40
13.2.5 Modifying properties ... 40
13.2.6 Associations ... 40
13.2.7 Methods ... 40
13.3 IANet_VLAN ... 40
13.3.1 Purpose ... 40
13.3.2 Instances ... 40
13.3.3 Creating instances ... 40
13.3.4 Removing Instances ... 40
13.3.5 Modifying properties ... 40
13.3.6 Associations ... 40
13.3.7 Methods ... 41
14. Getting the current configuration ... 42
14.1 Getting the Physical Adapters ... 42
14.2 Getting the PCI Devices ... 42
14.3 Getting the Adapter Settings ... 43
14.4 Getting the team configuration ... 43
14.5 Getting the team settings ... 44

14.6 Getting the VLAN configuration 45
14.7 Getting the VLAN settings 45
14.8 Getting the IP Protocol Information 46
14.9 Getting the Boot Agent Information 46
14.10 Getting the BootAgent Settings 46
15. Updating the configuration 48
15.1 Changing the adapter, team or VLAN settings 48
15.2 Creating a new (empty) team 48
15.3 Adding an adapter to a team 49
15.4 Removing an adapter from a team 49
15.5 Deleting a team 49
15.6 Changing the mode of a team 49
15.7 Changing an adapter's priority within a team 50
15.8 Uninstalling an adapter 50
15.9 Creating a VLAN 50
15.10 Changing the attributes of a VLAN 50
15.11 Deleting a VLAN 50
15.12 Updating the Boot Agent 51
16. Event Notifications 52
16.1 Event Schema 52
16.2 Concrete Event Classes 52
16.3 IANet_802dot3AdapterEvent 52
16.3.1 Purpose 52
16.3.2 Triggers 52
16.3.3 Event Data 52
16.4 IANet_802dot3TeamEvent 52
16.4.1 Purpose 53
16.4.2 Triggers 53
16.4.3 Event Data 53
16.5 IANet_802dot3VlanEvent 53
16.5.1 Purpose 53
16.5.2 Triggers 53
16.5.3 Event Data 53
16.6 Registering For Events 53
17. Optimized WQL Queries 54
17.1 Settings 54
17.1.1 Getting the settings for a particular adapter, VLAN or Team 54
17.1.2 Getting one setting 54
18. Summary of CIM classes 55
IANet_802dot1QVLANService 55
IANet_AdapterConfiguration 55
IANet_NetworkVirtualAdapter 58
IANet_SettingMultiSelection 59
IANet_SettingString 59
IANet_TeamedMemberAdapter 59
IANet_TeamOfAdapters 59
IANet_VLAN 59

1. Introduction

1.1 Scope

Network Configuration Services (NCS) is a means of deploying and managing all Intel end-station networking technologies using industry standard methods. This document describes the external view of the Intel® PRO Network Adapters WMI Provider (referred to throughout this document as "NCS WMI Provider").[1] The NCS WMI Provider is a network configuration block of NCS.

The NCS WMI Provider is a set of software components that implements the Intel WMI network classes. These classes are based on the Desktop Management Task Force (DMTF) CIM Schema version 2.5.

This document does not repeat information contained in the Managed Object Format (MOF) files provided with this product (e.g., details of the meanings of individual attributes can be found in the MOF attribute descriptions).

This document describes how a WMI application such as Intel® PROSet uses classes to configure a system's network. Readers should be familiar with WMI APIs and the WMI SDK (available from http://www.microsoft.com/).

1.2 Related Documents

- CIM schema version 2.0, 2.2 published by Desktop Management Task Force (DMTF), <http://www.dmtf.org>.
- Microsoft Windows Management Interface (and other manageability information) <http://www.microsoft.com/hwdev/WMI/>.
- Web-Based Enterprise Management (WBEM) initiative by DMTF <http://www.dmtf.org/wbem/index.html>.
- WMI (Microsoft CIM implementation) SDK <http://msdn.microsoft.com/code/sample.asp?url=/msdn-files/027/001/566/msdncompositedoc.xml>

[1] WMI stands for Windows Management Interface

2. WMI

Web-Based Enterprise Management (WBEM) is a Desktop Management Task Force (DMTF) initiative intended to provide enterprise system managers with a standardized, cost-effective method for end station management. The WBEM initiative encompasses a multitude of tasks, ranging from simple workstation configuration to full-scale enterprise management across multiple platforms. Central to the initiative is the Common Information Model (CIM), an extensible data model for representing objects that exist in typical management environments, and the Managed Object Format (MOF) language for defining and storing modeled data.

Windows Management Instrumentation (WMI) is an implementation of the WBEM initiative for Microsoft* Windows* platforms

WMI consists of two main components: the Core and the SDK.

Core - These components are part of the Operating System. They are required for a WMI-enabled application to work, and must be installed in order to use the SDK.

SDK - The SDK contains tools to browse the WMI schema, extend the schema, create providers, register and use WMI events. It also provides documentation useful in developing applications that will use WMI. The SDK is installed as part of the Microsoft Platform SDK installation process.
The SDK is supported on Windows NT4 SP4 or SP5, Windows 2000, Windows Me, Windows XP and Windows .NET Server.

The WMI architecture consists of the following components:

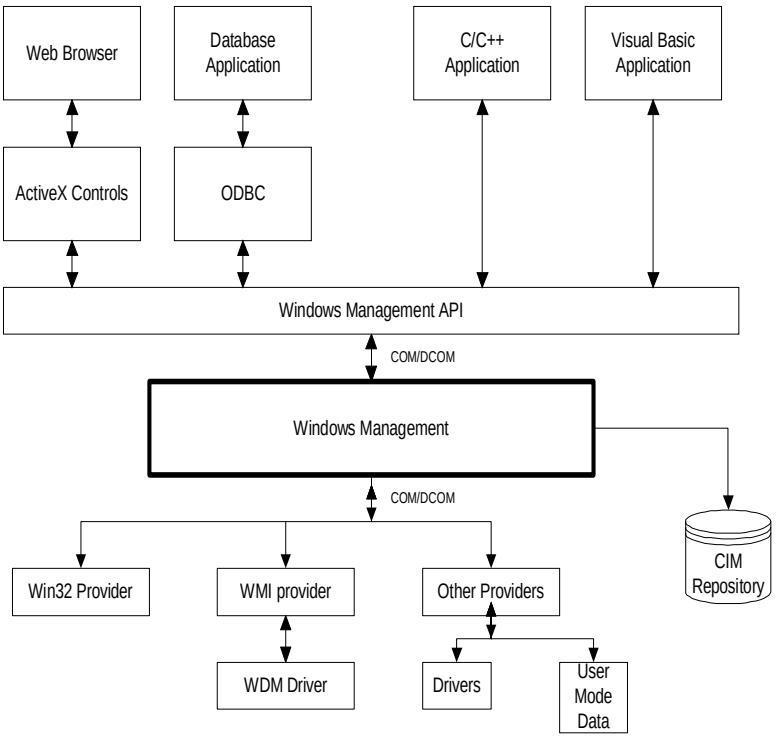
- Management applications
- Managed objects
- Providers
- Management infrastructure (consisting of the Windows Management and Windows Management repository)
- Windows Management API (which uses COM/DCOM to enable providers and management applications to communicate with the Windows Management infrastructure.

Management applications process or display data from managed objects, which are logical or physical enterprise components. These components are modeled using CIM and accessed by applications through Windows Management. Providers use the Windows Management API to supply Windows Management with data from managed objects, to handle requests from applications and to generate notification of events.

The management infrastructure consists of Windows Management (for handling the communication between management applications and providers) and the Windows Management repository (for storing data). The Windows Management repository holds static management data. Dynamic data is generated only on request from the providers. Data is placed in repository using either MOF language compiler or the Windows Management API.

Applications and providers communicate through Windows Management using the Windows Management API, which supplies such services as event notification and query processing.

The following diagram shows the interrelationship of these components:



2.1 Common Information Model (CIM Schema)

The Common Information Model (CIM) presents a consistent and unified view of all types of logical and physical objects in a managed environment. Managed objects are represented using object-oriented constructs such as classes. The classes include properties that describe data and methods that describe behavior. The CIM is designed by the DMTF to be operating system and platform independent, however the Microsoft implementation predominates the specification. The WBEM technology includes an extension of the CIM for the Microsoft Windows operating system platforms. Please refer to the DMTF CIM schema on DMTF web site for more information.

The CIM defines three levels of classes:

- Classes representing managed objects that apply to all areas of management. These classes provide a basic vocabulary for analyzing and describing managed systems and are part of what is referred to as the core model.
- Classes representing managed objects that apply to a specific management area but are independent of a particular implementation or technology. These classes are part of what is referred to as the common model - an extension of the core model.
- Classes representing managed objects that are technology-specific additions to the common model. These classes typically apply to specific platforms such as UNIX or the Microsoft Win32 environment.

All classes can be related by inheritance, where a child class includes data and methods from its parent class. Inheritance relationships are not typically visible to the management application using them, nor are the applications required to know the inheritance hierarchy. Class hierarchies can be obtained using applications that are included in the WMI Tools (see the WMI Tools at http://www.microsoft.com for more information).

Windows Management also supports association classes. Association classes link two different classes to model a user-defined relationship, and are visible to management applications. Windows Management defines association classes to support system classes. Third-party developers can also define association classes for their management environment.

WBEM supports the concept of schemas to group the classes and instances that are used within a particular management environment. The Platform SDK includes two schemas: the CIM schema and the Microsoft Win32 schema. The CIM schema contains the class definitions for the first two levels of the CIM. These classes represent managed objects that are part of every management environment regardless of platform. The Win32 schema contains class definitions for managed objects that are part of a typical Win32 environment.

For additional information on CIM, visit http://www.dmtf.org.

3. Main Features

The main features exposed by the NCS WMI Provider are divided into the following categories:

3.1 Implemented Features

3.1.1 Adapter

- Enumerate all physical adapters supported by Intel® PROSet.
- Enumerate an installed adapter's settings.
- Add/Remove/Update settings for an installed adapter.
- Obtain an adapter's Physical Device information.
- Obtain an adapter's System Slot Device information.
- Obtain the IPv4 protocol settings for an adapter.
- Uninstall an adapter.
- Update and change an adapters Boot Agent and associated settings.

3.1.2 Team

- Enumerate the teams supported by Intel® PROSet.
- Create/Remove a Team of adapters.
- Add/Remove/Update Settings of the Team.
- Add/Remove member adapters for a team.
- Obtain the IPv4 protocol settings for a team.

3.1.3 VLAN

- Enumerate Virtual LANs on an adapter or team.
- Create/Remove Virtual LANs on a physical adapter or a team of adapters.
- Add/Remove/Update Settings of the VLAN.
- Obtain the IPv4 protocol settings for a team.

3.1.4 Event Notification

- Allow the client to register for the following events:
    - Adapter status events
    - Adapter configuration events
    - Session events
    - Team status events
    - Team configuration events
    - VLAN configuration events

4. Installed Files

4.1 Executables

There are three separate executables for the Provider:

NcsWmiCo.exe	The core provider. Implements the IANet_NetService and the core event classes.
NcsWmiIm.exe	The instance and method provider. Implements the Ethernet Adapter Schema, the Teaming Schema, the Setting Schema and the VLAN Schema.
NcsWmiEv.exe	The event provider. Implements the adapter, team and VLAN events.

4.2 MOF files

There are separate MOF files for language neutral and language specific data. Also, there are separate MOF files for IntelNCS and CIMV2 namespace.

Refer to sections 7 and 8 for more details on this.

4.2.1 MOF files for IntelNCS namespace:

NcsCmLn.mof	CIM base classes on which the NCS classes depend.
NcsCmEnu.mfl	US English version of the CIM base classes
NcsCoLn.mof	Core classes implemented by the core provider.
NcsCoEnu.mfl	US English textual amendments to the core classes.
NcsIaLn.mof	Classes for the IEEE 802.3 adapters, teams and VLANs.
NcsIaEnu.mfl	US English textual amendments to the 802.3 core classes.

4.2.2 MOF files for CIMV2 namespace:

C2CmLn.mof	CIM base classes on which the NCS classes depend.
C2CmEnu.mfl	US English version of the CIM base classes
C2CoLn.mof	Core classes implemented by the core provider.
C2CoEnu.mfl	US English textual amendments to the core classes.
C2IaLn.mof	Classes for the IEEE 802.3 adapters, teams and VLANs.
C2IaEnu.mfl	US English textual amendments to the 802.3 core classes.

4.3 Resource Files

ENIU_8023.dll	English USA 8023 Resource.
ENU_NWRC.dll	English USA NCS WMI Resources for Core Providers.
ENU_NWR.dll	English USA NCS WMI Resources for 8023 Providers.

The Other localized resource files can be loaded on demand. The general pattern for the names of localized resource DLLs is “<code>_mwr.dll” where <code> is the language code for the localization (e.g. FRA for standard French).

5. Security

The NCS WMI Provider uses client impersonation to manage the security. Every call into the Provider will be made in the client's own security context. This context is passed down to the lower layers. An operation may fail if the user does not have suitable administrative rights on the target machine.

6. Namespace and Context

6.1 Namespace

The CIM classes reside in a namespace. The standard Microsoft namespace is called “root/cimv2” and is based on CIM v2,2. The NCS WMI Provider classes can be added to this namespace.

The NCS WMI Provider is based on CIM v2.5. Because of this, and because of differences used in the keys of the objects, the NCS WMI Provider classes are also in a separate namespace: “root/IntelNCS”. Intel® PROset uses the “root/IntelNCS” namespace.

6.2 WBEM Context

Context objects are used to provide additional information to the NCS WMI Provider that cannot be passed as a parameter to a WMI API method. Use the **IWbemContext** to register context qualifiers. The interface pointer for the context object is passed as the last parameter of an **IWbemServices** method.

The following table contains the context qualifiers (named values) used by the NCS WMI Provider. Most qualifiers, like **SessionHandle**, are only used in conjunction with specific functional areas of the Provider, whereas **LocaleID**, **MachineName** and **ApplicationName** can be set for all **IWbemServices** calls.

If no context is passed to the Provider, it will use the LocaleID passed in the **Initialize** call to the Provider.

Any Read done with a context will read the current configuration until a write operation is performed. Subsequent reads will show the system as it would be after the write has succeeded.

A NULL context can be used for reads.

Context Qualifier	Variant Type	Description
SessionHandle	VT_BSTR	Identifies the application's copy of IANet network classes. The application cannot make any changes to the classes or their attributes without first establishing a session handle. See the section on the IANet_NetService class to see how to establish and use a session handle. This qualifier is not required if the application is only going to read data from the classes. The session handle allows the NCS software to manage simultaneous multiple accesses to the configuration without one user locking out all others. Each session has a separate cache to store any changes that have been made. If there are multiple users making changes simultaneously then the first user to apply their changes will succeed. All the other users’ caches will be invalidated.
LocaleID	VT_BSTR	Microsoft's ID for a locale. This is required if the application requires localized text strings from the Provider. All error messages and warnings will be in English unless the required LocaleID is used.
ApplicationName	VT_BSTR	The name of the application that made the call. This is required for logging.
MachineName	VT_BSTR	The name of the machine that is connecting to the Provider. This is required for logging.
PreCheck	VT_BOOL	This Boolean value is used to tell the Provider that the client is attempting to verify that an operation is allowed before actually performing the operation. (For example, adding an adapter to a team). Values: • TRUE – Provider will not perform the operation, but will return an error code and extended status if the operation is not allowed • FALSE – Provider will perform the operation. If this qualifier is missing it has the same effect as if the attribute was FALSE.
WarningErrorCode	VT_I4	Some operations may require warnings to be sent to the user. (e.g., adding an adapter to the team may require the team to be reloaded in some circumstances). WMI does not

Context Qualifier	Variant Type	Description
		provide a mechanism for doing this. If this qualifier is present and non-zero then the Provider will return E_FAIL if the operation succeeded, but there was an associated warning. The client should use the extended status to get the text of the warning.

7. Locales and Localization

There are two aspects to Provider localization:

7.1 Localized MOF files

All the MOF files used by the NCS WMI Provider are localized according to the Microsoft® Windows® Management Instrumentation (WMI) globalization model. To accomplish this, each class definition is separated into the following:

- a language-neutral version that contains only the basic class definition in the **.mof** file.
- a language-specific version that contains localized information, such as property descriptions that are specific to a locale in the corresponding **.mfl** file.

7.1.1 Class Storage

The language-specific class definitions are stored in a child sub-namespace beneath the namespace that contains a language-neutral basic class definition. For example, for the NCS WMI Provider, a child namespace ms_409 will exist beneath the root/intelncs namespace for the English locale. Similarly, there exists a child sub-namespace for each supported language beneath the root/intelncs namespace.

7.1.2 Localized MOF support in cimv2 namespace

In case of root/cimv2 namespace the NCS WMI Provider classes (i.e., IANet_ classes) are derived from base classes added to this namespace by WMI. A child sub-namespace with language-specific class definitions for base classes should pre-exist beneath the root/cimv2 namespace. The IA_Net language-specific class definitions will be added to this existing child namespace. Due to this dependency on base classes, MOF localization is done only in the default system locale.

7.1.3 Runtime support

To retrieve localized data, a WMI application can specify the locale using *strLocale* parameter in **SWbemLocator.ConnectServer** and **IWbemLocator::ConnectServer** calls. If the locale is not specified, the default locale for that system will be used. (e.g. MS_409 for US English). This locale is used to select the correct namespace when adding in the English strings.

In addition, **IWbemServices::GetObject**, **SwbemServices.GetObject**, **IWbemServices:: ExecQuery**, and **SWbemServices.ExecQuery** must specify the WBEM_FLAG_USE_AMENDED_QUALIFIERS flag to request localized data along with the basic definition. This is required in all functions that produce displayable values using value maps or display descriptions or other amended qualifiers from the MOF files.

7.2 Localized attribute data

To get localized attribute data (such as error messages), the NCS WMI Provider needs to know the locale of the caller for every call. For this to work correctly the client must add the locale to the context object that is passed for every call (see section 6 on WBEM Context). If the Provider needs to return a localizable string, then it will attempt to load a resource DLL that is appropriate for the client's locale. If there is no appropriate resource DLL, then the Provider will return strings in US English.

8. Error Reporting

8.1 IANet_ExtendedStatus

This section details how to handle errors generated by NCS WMI Provider.

How and when an error object is returned depends on whether a call is synchronous, semi-synchronous or asynchronous. In most cases, the HRESULT is set to WBEM_E_FAILED when an error occurs. At this point, however, it is unknown whether WMI or the NCS WMI Provider generated the error.

8.1.1 Getting the error object

For synchronous calls:

Use GetErrorInfo() to get the IErrorInfo object. Use QueryInterface() to get the IWbemClassObject that contains the error information.

For asynchronous calls:

The IWbemClassObject is passed back as the last item in the last SetStatus() call.

After you get the error object instance, you can check the __**Class** property to determine the origin of the error. WMI creates an instance of __**ExtendedStatus,** and the NCS WMI Provider creates an instance of **IANet_ExtendedStatus** for errors relating to IANet_ classes and NCS WMI Provider.

IANet_ExtendedStatus is derived from __**ExtendedStatus** and contains the following attributes:

8.1.2 Error Object Qualifiers

Context Qualifier	Description
Description	Description of the error tailored to the current locale
File	Code file where the error was generated
Line	Line number of the code file with the error
ParameterInfo	Class or attribute that was being utilized when the error occurred
Operation	Operation being attempted when the error occurred
ProviderName	Name of the Provider that caused the error
StatusCode	Code returned from the internal call that failed
SessionHandle	Session handle used for the operation
RuleFailureReasons	Reason for operation failure. An operation can fail because a technical rule has failed. (e.g., you must have a management adapter in certain teams).

8.2 Error Codes

For all error codes, the NCS WMI Providers gives a description customized to the locale. Below is a list of possible error codes that the Provider may return. Error codes are in the form of HRESULT with severity set to one (1) and facility set to ITF. An application may use these codes as a basis for a recovery action.

0x80040901 "WMI: Put property failed"
0x80040902 "WMI: No class object"
0x80040903 "WMI: Failed to create class"
0x80040904 "WMI: Failed to spawn instance of class"
0x80040905 "WMI: Failed to create safe array"
0x80040906 "WMI: Failed to put safe array"
0x80040907 "WMI: Failed to return object to WMI"
0x80040908 "WMI: Get property failed"
0x80040909 "WMI: Unexpected type while getting property"
0x8004090A "WMI: Class not implemented by this provider"
0x8004090B "WMI: Unable to parse WQL statement"
0x8004090C "WMI: Provider only supports WQL"
0x8004090D "WMI: Parameter in context has the wrong type"
0x8004090E "WMI: Error formatting debug log"
0x8004090F "WMI: bad object path"
0x80040910 "WMI: Failed to update setting"
0x80040911 "WMI: Null parameter passed to method"
0x80040912 "Setting value too small."
0x80040913 "Setting value too big."

0x80040914 "Setting not in step"
0x80040915 "String setting is too long"
0x80040916 "Setting is not one of the allowed values"
0x80040917 "WMI: Qualifier not found"
0x80040918 "WMI: Qualifer set not found"
0x80040919 "WMI: Safe array access failed"
0x8004091A "WMI: Unhandled exception"
0x8004091B "WMI: Operation is not supported for this class"
0x8004091C "WMI: Unexpected event class"
0x8004091D "WMI: Bad event data"
0x8004091E "WMI: Operation succeeded with warnings"
0x8004081F "WMI: The NCS Service has been stopped."

0x80040801 "EAL: Internal exception"
0x80040802 "EAL: General failure"
0x80040803 "EAL: Not initialized"
0x80040804 "EAL: Failed to intialize."
0x80040805 "EAL: Session limits exceeded"
0x80040806 "EAL: Out of memory"
0x80040807 "EAL: Rule syntax error"
0x80040808 "EAL: Unexpected end of list"
0x80040809 "EAL: Rule link error"
0x8004080A "EAL: Device Creation Failed"
0x8004080B "EAL: Media service not found"
0x8004080C "EAL: Device service not found"
0x8004080D "EAL: PCI bus module not found"
0x8004080E "EAL: Adapter is a member of a team"
0x8004080F "EAL: Rule Access Point creation error"
0x80040810 "EAL: Registry key error"
0x80040811 "EAL: Registry XML file path error"
0x80040812 "EAL: Unknown event class"
0x80040813 "EAL: Unknown module id"
0x80040814 "EAL: Rule service not found"
0x80040815 "EAL: NULL input pointer"
0x80040816 "EAL: Rule grammar error"
0x80040817 "EAL: Rule failed"
0x80040818 "EAL: Setting is already grouped"

0x80040220 "Sync Layer: Team removal failed."
0x80040221 "Sync Layer: Vlan creation failed."
0x80040222 "Sync Layer: Vlan removal failed."
0x80040223 "Sync Layer: Adapter removal failed."
0x80040224 "Sync Layer: Setting Change/Creation/Removal failed."
0x80040225 "Sync Layer: Parameter Change/Removal failed."
0x80040226 "Sync Layer: NetConfig subsystem locked. "
0x80040227 "Sync Layer: System Update In Progress. Please try again later."
0x80040228 "Sync Layer: Adapter is Locked"
0x80040229 "Sync Layer: Flash read failed."
0x8004022A "Sync Layer:"

0x80040210 "Sync Layer: Invalid event offset"
0x80040211 "Sync Layer: Invalid input"
0x80040212 "Sync Layer: Invalid key"
0x80040213 "Sync Layer: Adapter not team member"
0x80040214 "Sync Layer: Driver not loaded"
0x80040215 "Sync Layer: Client impersonation failed"
0x80040216 "Sync Layer: Caught exception"
0x80040217 "Sync Layer: Session not locked"
0x80040218 "Sync Layer: Hardware access layer is not available"
0x80040219 "Sync Layer: Flash not available"
0x8004021A "Sync Layer: Diagnostics not supported"
0x8004021B "Sync Layer: Diagnostic test not running"
0x8004021C "Sync Layer: Boot Agent update not available"
0x8004021D "Sync Layer: Boot Agent corrupted."
0x8004021E "Sync Layer: Flash write failed."
0x8004021F "Sync Layer: Team creation failed."
0x80040201 "Sync Layer: Initialization failed"
0x80040202 "Sync Layer: Invalid initialization handle"
0x80040203 "Sync Layer: Session handle already exists"
0x80040204 "Sync Layer: Invalid session handle"

0x80040205 "Sync Layer: The maximum number of sessions has been reached."
0x80040206 "Sync Layer: The session lock handle already exists"
0x80040207 "Sync Layer: Invalid session lock handle"
0x80040208 "Sync Layer: Session already locked"
0x80040209 "Sync Layer: Invalid media service module Id"
0x8004020A "Sync Layer: Invalid Advanced Service Module Id"
0x8004020B "Sync Layer: Invalid device service module Id"
0x8004020C "Sync Layer: Invalid component type Id"
0x8004020D "Sync Layer: Invalid bus interface module Id"
0x8004020E "Sync Layer: Invalid sink window handle"
0x8004020F "Sync Layer: Invalid event Id"

0x80040401 "HAM PCI: Invalid memory map address"
0x80040402 "HAM PCI: Configuration driver failed to load"
0x80040403 "HAM PCI: Configuration driver version mismatch"
0x80040404 "HAM PCI: Device slot not found"
0x80040405 "HAM PCI: Diagnostic driver failed to load"
0x80040406 "HAM PCI: Diagnostic driver version mismatch"
0x80040407 "HAM PCI: Diagnostic driver initialization failed"
0x80040408 "HAM PCI: Diagnostics not initialized"
0x80040409 "HAM PCI: Diagnostics already initialized"
0x8004040A "HAM PCI: Diagnostic test already running"
0x8004040B "HAM PCI: Diagnostic test not running"
0x8004040C "HAM PCI: Diagnostic test terminated"
0x8004040D "HAM PCI: Diagnostic Invalid test number"
0x8004040E "HAM PCI: Diagnostic hardware missing"
0x8004040F "HAM PCI: Diagnostic send receive initialization failed"

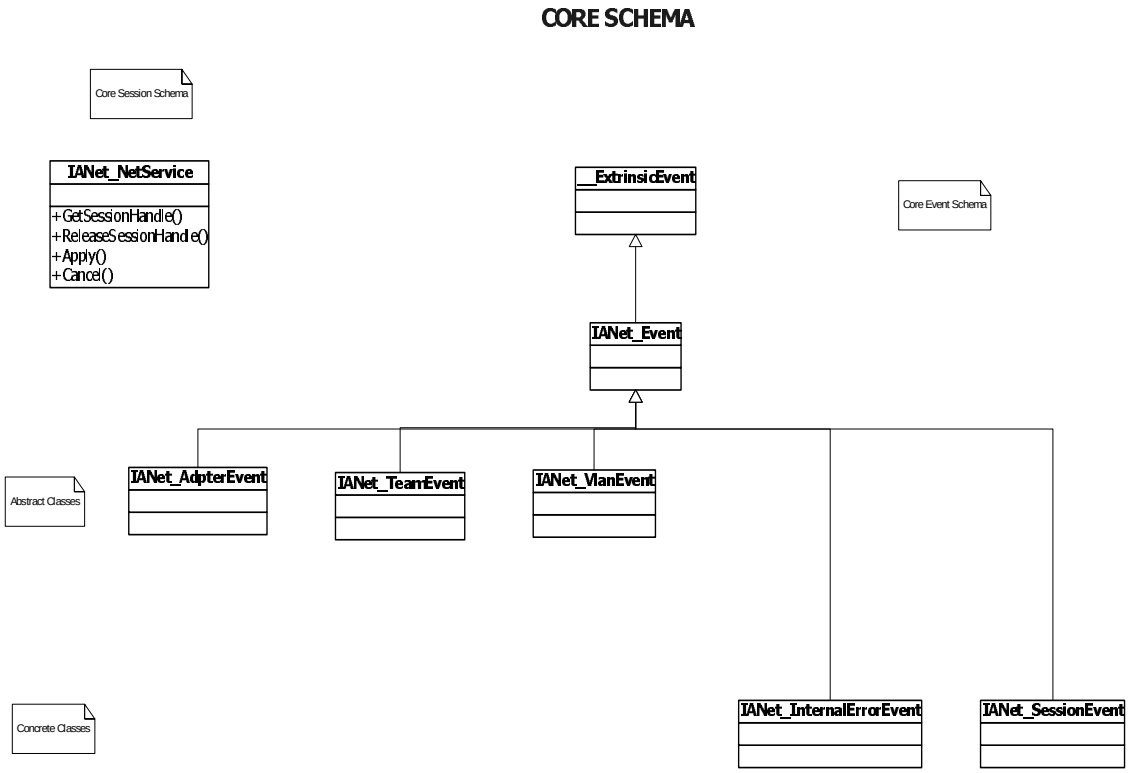
0x80040511 "Media Service: NDIS IO call failed"
0x80040512 "Media Service: Miniport not loaded"
0x8004051B "Media Service: Invalid device handle"
0x8004051C "Media Service: Invalid adapter handle"
0x8004051D "Media Service: Invalid team handle"
0x8004051E "Media Service: Invalid VLAN handle"
0x8004051F "Media Service: Device missing"
0x80040520 "Media Service: Invalid setting type"
0x80040521 "Media Service: Unknown invalid object"
0x80040522 "Media Service: Invalid Setting Handle"
0x80040523 "Media Service: Invalid Team Mode"
0x80040525 "Media Service: Setting Already Exists"

0x80042001 "RAP: Already initialized"
0x80042002 "RAP: Invalid XML file"
0x80042003 "RAP: XML load error"
0x80042004 "RAP: Not initialized"
0x80042005 "RAP: Rule not extracted before"
0x80042006 "RAP: Conditions count mismatch"
0x80042007 "RAP: Results apply error"
0x80042008 "RAP: Invalid rule"
0x80042009 "RAP: Node not found"
0x8004200A "RAP: Error no single node"
0x8004200B "RAP: No action rule"
0x8004200C "RAP: Zero condition"
0x8004200D "RAP: Zero action"
0x8004200E "RAP: XML Decode error"

9. The Core Schema

The Core Schema consists of the IANet_NetService class and the core event classes.

9.1 Schema Diagram



9.2 IANet_NetService

9.2.1 Purpose

The IANet_NetService class is the root object from the IANet_ schema. This class enables the client to access the sessions that are required to perform sets.

9.2.2 Instances

There is one instance of this object. The client should not rely on the key used for this class. Instead, the client should get the instance of the class by enumerating all instances of IANet_NetService.

9.2.3 Creating instances

The user is not able to create instances of IANet_NetService.

9.2.4 Removing instances

The user is not able to delete the instance of IANet_NetService.

9.2.5 Modifying properties

There are no user modifiable properties of this class.

9.2.6 Supported Attributes

This class implements two attributes:

- Version. contains the current version of the Core Provider
- InstallDate. contains the date that the providers were installed

9.2.7 Methods

There are several methods for this class used to manage sessions:

void GetSessionHandle(([OUT] string SessionHandle, [out] uint32 ActiveSessions)	Used to set a session handle string, which should be placed in the context object in the SessionHandle qualifier. ActiveSessions returns the number of active sessions for this system. (this allows clients to warn that others may be modifying the network settings).
void Apply([IN] string sSessionHandle, [OUT] uint32 FollowupAction);	Applies changes made with a particular session handle. The uint32 argument returned is used by the Provider to tell the application the server must be rebooted before the changes will take effect. (this can be accomplished by calling the Reboot method on the class Win32_OperatingSystem). Values: 1 – system reboot required 0 – no reboot required
void ReleaseSessionHandle ([IN] string SessionHandle)	Releases a session handle after it has been used. Any changes done with this session will be lost. The session handle will no longer be valid after this call, and can no longer be used.
void Cancel([IN] string SessionHandle);	Cancels a session. The internal cache will be cleared, and any data read following this call will show the current configuration.

9.3 Core Events

9.3.1 IANet_SessionEvent

9.3.2 Purpose

This event is used to notify the client about the use of the NCS session API. Clients can use this event to be informed if other clients are creating or using sessions.

9.3.3 Triggers

This event is triggered whenever a client creates a session, deletes a session, or calls apply for a session.

9.3.4 Event Data

The **EventType** can take one of the following values:

- “New session”. A new session has been created. The session may have been created by the client itself or another client.

- “End session”. A client has finished with a session. The session may have been ended by the client itself or another client.
- “Cache invalidated”. Another client has called “Apply” on a session. All other sessions are invalided and cache associated with their sessions has been deleted.
- “Configuration changed”. Configuration for this session has changed.

The **SessionHandle** contains the session handle that triggered the event.

OpenSessions contains the number of open sessions. This data item is NULL for the “Cache invalidated” and “Configuration changed” events.

9.3.4.1 IANet_InternalErrorEvent

9.3.5 Purpose

This event is used to notify the client that an internal error has occurred in the event Provider. In some cases this means that the event provider is not able to report further events.

9.3.6 Triggers

The event will occur:

- after the event provider gets an unknown event from an event source, or;
- after the software that provides the events has been shutdown, or;
- after the event provider gets an event but the event source cannot get further data about the event.

9.3.7 Event Data

The **EventType** can be one of the following:

- "Could not get event data". An event occurred but the event source cannot get further data about the event.
- "Event source has shut down". The data source for the event was shutdown. In this case, the event provider will also be shutdown and no more events will be generated until the source is restarted and new notification queries are made.
- "Unexpected message". The event provider received an unexpected event.

9.4 Use Cases

A session handle is required to change the configuration. The session handle allows the NCS software to manage simultaneous multiple accesses to the configuration, thereby preventing a session from locking out all others. Each session has a separate cache to store any changes that have been made. If there are multiple sessions making changes simultaneously then the first to apply its changes will succeed. All other session caches will be invalidated.

9.4.1 Getting a Session Handle

The client must get the object path of the single instance of IANet_NetService before accessing the session handle. Call IWbemServices::CreateInstanceEnum and pass the name of the class: IANet_NetService. (this is equivalent to calling IWbemServices::ExecQuery with the query “SELECT * FROM IANet_NetService).

Before making any changes to the configuration, the client must get a session handle. Use the **GetSesssionHandle** method to start a fresh session. The client can use IWbemServices::ExecMethod to execute a method on a CIM object and will need the object path, from **__PATH** attribute of the instance of IANet_NetService.

This method also returns the number of currently active sessions. The client may wish to issue a warning if it does not have exclusive access to the Network Configuration Service (NCS).

9.4.2 Using a session handle in the IWbemContext object

After the client obtains a session handle, it must create an IWbemContext object. Store the session handle in the **SessionHandle** qualifier of this object. A pointer to this COM object should be passed to every call into IWbemServices. The session handle is not required when making calls to access the IANet_NetService object as this takes the handle as an explicit argument.

9.4.3 Reading pending changes using a session handle

When reading the configuration, if the user passes the session handle in the context, then the Provider will return the configuration as if the pending updates were applied (e.g., uninstalled adapters will be missing and changed settings will return their new values).

However, some objects will not appear until the apply has been called (e.g., IANet_IPProtcolEndpoints will not be created until the protocol has been bound to the appropriate miniport).

9.4.4 Finishing with a session handle

After changing the configuration, call the **Apply** method to commit the changes. This may return a follow-up action code (e.g., reboot the system before the changes can take effect).

Always call the **ReleaseSessionHandle** after a session is finished – otherwise any changes made will be discarded.

Calling the **Cancel** method will also discard any changes made, but the client can continue to use the session handle as if it has just been created.

9.4.5 Registering for the Core Events

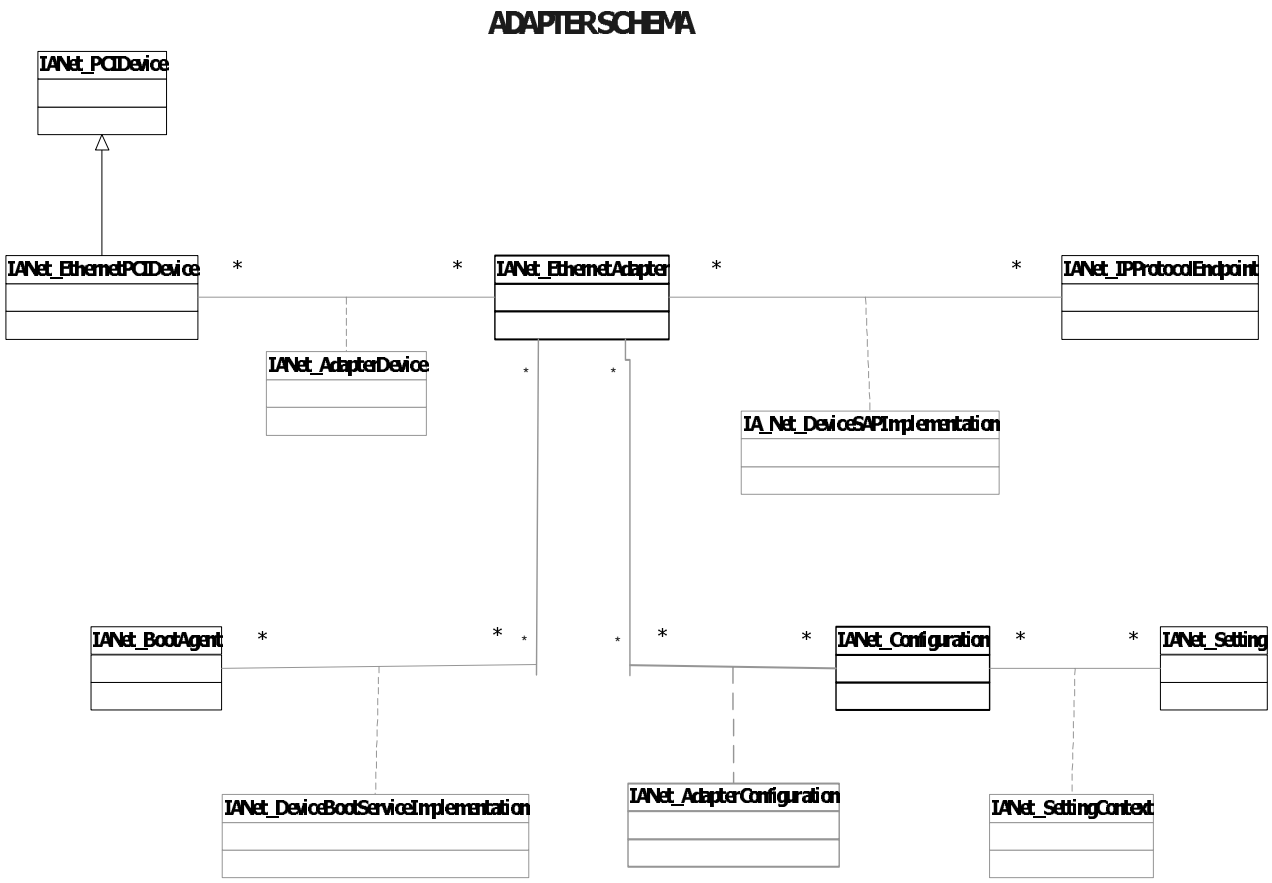
Applications should use IWbemServices::ExecNotificationQuery or use IWbemServices:: ExecNotificationQueryAsync to request event notification. The following queries are examples of event notification queries (this list is not exhaustive as many queries are possible).

Task	Notification Query
Request all events	SELECT * FROM IANet_Event
Request all session events	SELECT * FROM IANet_SessionEvent
Request all internal events	SELECT * FROM IANet_InternalErrorEvent

10. Ethernet Adapter Schema

The adapter schema is used to model the various configurable Intel® PROSet Ethernet Adapters. This schema is based on the CIM v2.5 schema.

10.1 Schema Diagram



10.2 IANet_EthernetAdapter

10.2.1 Purpose

IANet_EthernetAdapter defines the capabilities and status of all the installed Intel/PRO® Network Adapters as well as any other adapters that can be teamed using Intel intermediate drivers. The class is derived from the CIM_EthernetAdapter superclass defined in CIMv2.5. CIM_EthernetAdapter is derived from CIM_NetworkAdapter, an Abstract class defining general networking hardware concepts such as PermanentAddress, CurrentAddress, Speed of operation, etc.

10.2.2 Instances

Instances of this class will exist for each of the following:

- supported and installed Intel NICs
- non-Intel NICs that can participate in an Intel multi-vendor team
- created Intel adapter team

10.2.3 Creating instances

The user cannot create instances of IANet_EthernetAdapter.

10.2.4 Removing instances

Deleting an instance of IANet_EthernetAdapter will uninstall physical adapters. Only non-virtual Intel adapters may be uninstalled in this way. A session handle is required for this operation.

10.2.5 Modifying properties

There are no user-modifiable properties for this class.

10.2.6 Unsupported Attributes

The following attributes are not required for Intel® PROSet and are, therefore, not supported:

AutoSense (this is exposed as a setting), ErrorCleared, OtherIdentifyingInfo, IdentifyingDescriptions, InstallDate, LastErrorCode, MaxDataSize, MaxQuiesceTime, PowerManagementCapabilities (this is exposed as a method), PowerManagementSupported (this is exposed as a method), PowerOnHours, ShortFramesReceived, SymbolErrors, TotalPowerOnHours.

10.2.7 Methods:

This class instance supports following methods:

IdentifyAdapter	Identifies adapter by flashing the light on the adapter for a few seconds. This method will only work for physical adapters.
HasVLANs	Returns number of VLANs on this adapter.
IsPowerMgmtSupported	Indicates whether power management is supported on the adapter.
GetPowerUsage	Detect overall power usage for the adapter. 0 = Normal Power; 1 = Low power
SetPowerUsage	Reduces overall power usage for the adapter. Note: Power usage setting is not preserved between system restarts or driver re-loads. When the system restarts, or the driver is reloaded, the adapter automatically returns to normal power consumption.
GetPowerUsageOptions	Detects any optional power usage settings (e.g., power usage for standby, battery operation, etc.).
SetPowerUsageOptions	Changes power usage options (e.g., method can be used to reduce power usage for standby, battery operation, etc.) Note: Power usage settings are stored and used for subsequent reboots.
TestCable	Performs diagnostic test on a particular adapter. On failure, this method returns possible problems, causes and solutions.
AdvancedTestCable	Performs advanced cable tests on a particular adapter. This test suit is available with 1000 Mbps adapters. Method returns names of tests and their respective results. Note: Link failure can occur when "SpeedDuplex" is not set to "Auto Negotiate". In this instance, "bSpeedAndDuplexNotAutomatic" out parameter is TRUE.
TestLinkSpeed	Determines whether adapter is running at full speed. If the adapter is advertising less than 1 Gigabit, then method states possible reasons (e.g., "Link partner is not capable of running at 1000 Mbps").

10.3 IANet_IPProtocolEndpoint

10.3.1 Purpose

This class is used to describe the IP settings for a protocol endpoint in the system. The NCS WMI Provider does not provide information for any other types of networking protocol. The class is derived from the abstract class CIM_IPProtocolEndpoint. The NCS WMI Provider only supplies protocol information when it concerns an entity managed by Intel® PROSet.

10.3.2 Instances

An IANet_IPProtocolEndpoint instance will exist for each binding of the IP Protocol stack to an Intel supported endpoint. (i.e., Intel adapters, Intel team-able adapters and VLANs). Some teamed adapters do not have their own IP addresses and so have no IANet_IPProtocolEndpoint directly associated with their adapter instance.

The IANet_IPProtocolEndpoint exists only after the operating system has bound the protocol to the adapter or VLAN.

Though some adapters may have more than one IP address, they will be associated with only one IP Protocol Endpoint instance. The Provider does not support this advanced use as it is not required or used by Intel® PROSet.

10.3.3 Creating instances

The user cannot create instances of IANet_IPProtocolEndpoint. The instance exists only if the operating system has bound the protocol to the endpoint.

10.3.4 Removing instances

The user cannot remove instances of IANet_IPProtocolEndpoint

10.3.5 Modifying properties

There are no user-modifiable properties for this class.

10.3.6 Associations

An instance IANet_AdapterProtocolImplementation is used to associate an IANet_EthernetAdapter with an IANet_IPProtocolEndpoint.

An instance of IANet_VLANProtocolDependency is used to associate a VLAN with an IANet_IPProtocolEndpoint.

Note: Teams are associated with the endpoint via the adapter that represents the virtual adapter for the team.

10.3.7 Supported Attributes

The following read only attributes are required by Intel® PROSet:

Address, AddressType, DefaultGateway, DHCPServerAddress, DHCPAutoAssign, IPVersionSupport, SubnetMask

10.3.8 Unsupported Attributes

The following attributes are not required for Intel® PROSet and are, therefore, not supported:

Caption, Description, InstallDate, NameFormat, OtherTypeInformation, ProtocolType, Status.

10.3.9 Methods:

None.

10.4 IANet_BootAgent

10.4.1 Purpose

This class is used to capture the information about the network boot capabilities of an adapter. (e.g., settings for the PXE Boot Agent supported by some Intel adapters). This class is derived from CIM_BootService.

10.4.2 Instances

An IANet_BootAgent instance exists for each adapter that supports boot agent capability, even if the boot agent is not currently installed.

10.4.3 Creating instances

The user cannot create instances of IANet_BootAgent. An instance exists only if the adapter supports boot agent functionality.

10.4.4 Removing instances

The user cannot remove instances of IANet_BootAgent.

10.4.5 Modifying properties

There are no user-modifiable properties of this class.

10.4.6 Associations

An instance of IANet_DeviceBootServiceImplementation is used to associate an IANet_EthernetAdapter with an IANet_BootAgent, if the adapter supports it.

10.4.7 Supported Attributes

The following read only attributes are required by Intel® PROSet:

InvalidImageSignature, Version, UpdateAvailable, FlashImageType.

10.4.8 Unsupported Attributes

The following attributes are not required by Intel® PROSet and are, therefore, not supported:

Caption, Description, InstallDate, Started, StartMode, Status.

10.4.9 Methods

There are two methods on this class that can be used to update the Flash ROM on the NIC:

uint32 ProgramFlash([IN, ValueMap {"0","1"} , Values {"Check Version", "Write Flash"}: Amended] uint32 Action, [IN] uint8 NewFlashData[], [OUT] string strErrorMessage);	This method is used to update the Flash ROM on the NIC. This will cause the NIC to stop communicating with the network while the flash is updated.
uint32 ReadFlash([OUT] uint8 FlashData[]);	This method reads the Flash ROM on the NIC.

10.5 IANet_PCIDevice

10.5.1 Purpose

This class is used to describe the properties of a PCI Device for a network device in the system. The class is derived from CIM_PCIDevice.

10.5.2 Instances

An instance of this class exists for each PCI card that is a network device in the system (for IA64, only PCI devices that are Intel® PROSet supported adapters will have instances).

10.5.3 Creating instances

The user cannot create instances of IANet_PCIDevice.

10.5.4 Removing instances

The user cannot remove instances of IANet_PCIDevice.

10.5.5 Modifying properties

There are no user-modifiable properties of this class.

10.5.6 Associations

See IANet_EthernetPCIDevice for the class associations.

10.5.7 Methods

There are no supported methods for this class.

10.5.8 Unsupported Attributes

The following attributes are not supported by the NCS WMI Providers:

AdditionalAvailabitlity, Capabilities,,CapabilityDescriptions, Caption, DeviceSelectTiming, ErrorCleared, ErrorDescription, IdentifyingDescription, InstallDate, LastErrorCode, MaxNumberController, MaxQuiesceTime, Name, OtherIdentifyingInfo, PowerManagementCapabilities, PowerManagementSupported, PowerOnHours, ProtocolDescription, ProtocolSupported, SelfTestEnabled, TimeOfLastReset, TotalPowerOnHours.

10.6 IANet_EthernetPCIDevice

10.6.1 Purpose

This class is used to describe the properties of a PCI Device for a Intel® PROSet supported Ethernet adapter. This is a subclass of IANet_PCIDevice. The class contains some extra attributes that are only known for Intel® PROSet supported PCI devices.

10.6.2 Instances

An instance of this class exists for each PCI card that is a Intel® PROSet supported Ethernet adapter.

10.6.3 Creating instances

The user cannot create instances of IANet_EthernetPCIDevice.

10.6.4 Removing instances

The user cannot remove instances of IANet_ EthernetPCIDevice.

10.6.5 Modifying properties

There are no user-modifiable properties for this class.

10.6.6 Associations

An instance IANet_AdapterDevice is used to associate the IANet_PCIDevice with the IANet_EthernetAdapter. Virtual adapters (i.e., adapters created to represent teams) do not have an associated IANet_PCIDevice.

10.6.7 Unsupported Attributes

The following attributes are not supported by the NCS WMI Providers:

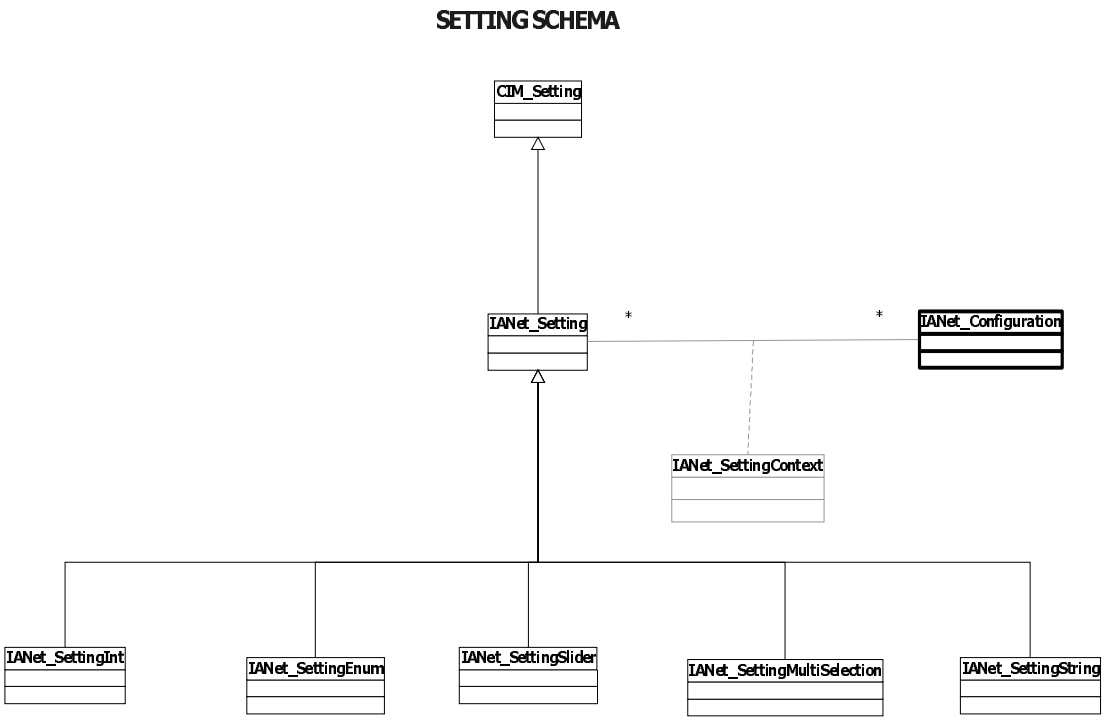
AdditionalAvailabitlity, Capabilities,,CapabilityDescriptions, Caption, DeviceSelectTiming, ErrorCleared, ErrorDescription, IdentifyingDescription, InstallDate, LastErrorCode, MaxNumberController, MaxQuiesceTime, Name, OtherIdentifyingInfo, PowerManagementCapabilities, PowerManagementSupported, PowerOnHours, ProtocolDescription, ProtocolSupported, SelfTestEnabled, Status, StatusInfo, TimeOfLastReset, TotalPowerOnHours.

10.6.8 Methods

There are no supported methods for this class.

11. Setting Schema

11.1 Schema Diagram



11.2 IANet_Configuration

11.2.1 Purpose

This class is used to group a collection of IANet_Setting instances. The class is derived from CIM_Configuration.

11.2.2 Instances

Each adapter, VLAN or Team can have several associated IANet_Configuration instances (each configuration corresponds to a different usage scenario for the adapter.

For this Provider release, there will be only one instance of IANet_Configuration per adapter, VLAN or team.

11.2.3 Creating instances

The user cannot create instances of IANet_Configuration.

11.2.4 Removing instances

The user cannot remove instances of IANet_Configuration.

11.2.5 Modifying properties

There are no user-modifiable properties for this class.

11.2.6 Associations

An IANet_AdapterConfiguration instance will exist to associate each adapter (IANet_EthernetAdapter) with its configuration.

An IANet_VLANConfiguration instance will exist to associate each VLAN (IANet_VLAN) with its configuration.

An IANet_BootAgentConfiguration instance will exist to associate each boot agent (IANet_BootAgent) with its configuration.

11.2.7 Methods

There are no supported methods for this class.

11.2.8 Unsupported Attributes

None.

11.3 IANet_Setting

11.3.1 Purpose

This abstract class is used to describe a settable property in a configuration. The class is derived from CIM_Setting.

11.3.2 Instances

A separate instance of this class will exist for each setting on each adapter, VLAN or Team. Settings are not shared between configurations.

There are several sub-classes for IANet_Setting. The sub-classes correspond to the different types and ranges of values that settings can take. Each sub-class corresponds to a different style of GUI that may be used to display or change the settings.

11.3.3 Creating instances

The user cannot create instances of IANet_Setting.

11.3.4 Removing instances

The user cannot remove instances of IANet_Setting.

11.3.5 Modifying properties

This abstract class has no modifiable properties, however, the child classes *have* modifiable properties (see below).

11.3.6 Associations

Each IANet_Setting instance is associated with an IANet_Configuration instance using an instance of IANet_SettingContext.

11.3.7 Methods

There are no supported methods for this class. To make changes to a setting, modify the required property and call PutInstance.

11.3.8 Unsupported Attributes

SettingID is not used.

11.4 IANet_SettingInt

11.4.1 Purpose

The class models a setting that takes an integer value. There are several IANet setting classes used to model integers. The differences between these classes concerns how the integer is displayed and modified by the GUI, and how validation is done by the Provider. For IANet_SettingInt, it is expected that the GUI will display an edit box with a spin control.

11.4.2 Instances

An instance of this class exists for each setting that should be displayed as an integer edit box.

11.4.3 Creating instances

The user cannot create instances of this class.

11.4.4 Removing instances

The user cannot remove instances of this class.

11.4.5 Modifying properties

The "CurrentValue" attribute is the only modifiable property of this class. The user can modify this property by using IWbemClassObject::Put() to change the value, then call "IWbemServices::PutInstance()" to update the setting. The Provider will check that:

CurrentValue <= **max**
CurrentValue >= **min**
(**CurrentValue** – **min**) is a multiple of **Step**

Where **max**, **min**, **CurrentValue** and **Step** are all attributes of IANet_SettingInt.

11.4.6 Associations

Each IANet_SettingInt instance is associated with an IANet_Configuration instance using an instance of IANet_SettingContext.

11.4.7 Unsupported Attributes

SettingID is not used.

11.4.8 Methods

There are no supported methods for this class. To make changes to a setting, modify the required property and call PutInstance.

11.5 IANet_SettingEnum

11.5.1 Purpose

The class models a setting that takes an integer value. There are several IANet setting classes used to model integers. The differences between these classes concerns how the integer is displayed and modified by the GUI, and how validation is done by the Provider. For IANet_SettingEnum, it is expected that the GUI will display a list of strings which map onto a small number of enumerated values. (e.g., a drop list combo box).

11.5.2 Instances

An instance of this class exists for each setting that will be displayed as an enum.

11.5.3 Creating instances

The user cannot create instances of this class.

11.5.4 Removing instances

The user cannot remove instances of this class.

11.5.5 Modifying properties

The "CurrentValue" attribute is the only modifiable property of this class. Modify this property by using Put() to change the value, then call "PutInstance()" to update the setting. The Provider will check that:

CurrentValue $\in$ **PossibleValues[]**

11.5.6 Associations

Each IANet_SettingEnum instance is associated with an IANet_Configuration instance using an instance of IANet_SettingContext.

11.5.7 Unsupported Attributes

SettingID is not used.

11.5.8 Methods

There are no supported methods on this class. To make changes to a setting modify the required property and call PutInstance.

11.6 IANet_SettingSlider

11.6.1 Purpose

The class models a setting that takes an integer value. There are several IANet setting classes used to model integers. The differences between these classes concerns how the integer is displayed and modified by the GUI, and how validation is done by the Provider. For IANet_SettingSlider, it is expected that the GUI will display a slider which will allow the user to choose the value in a graphical manner – the actual value chosen need not be displayed.

11.6.2 Instances

An instance of this class exists for each setting that will be displayed as a slider.

11.6.3 Creating instances

The user cannot create instances of this class.

11.6.4 Removing instances

The cannot remove instances of this class.

11.6.5 Modifying properties

The “CurrentValue” attribute is the only modifiable property of this class. Modify this property by using Put() to change the value, then call “PutInstance()” to update the setting. The Provider will check that:

CurrentValue $\in$ **PossibleValues[]**

11.6.6 Associations

Each IANet_SettingSlider instance is associated with an IANet_Configuration instance using an instance of IANet_SettingContext.

11.6.7 Unsupported Attributes

SettingID is not used.

11.6.8 Methods

There are no supported methods on this class. To make changes to a setting, modify the required property and call PutInstance.

11.7 IANet_SettingMultiSelection

11.7.1 Purpose

This class models a setting whereby the user can select several options from a list of options. For IANet_SettingMultiSelection, it is expected that the GUI will display multi-selection list box which will allow the user to choose any (or no) option(s).

11.7.2 Instances

An instance of this class exists for each setting that will be displayed as a multi-selection.

11.7.3 Creating instances

The user cannot create instances of this class.

11.7.4 Removing instances

The user cannot remove instances of this class.

11.7.5 Modifying properties

The “CurrentValue” attribute is the only modifiable property of this class. Modify this property by using Put() to change the value, then use “PutInstance()” to update the setting. The Provider will check that:

CurrentValue $\subseteq$ **PossibleValues[]**

11.7.6 Associations

Each IANet_SettingMultiSelection instance is associated with an IANet_Configuration instance using an instance of IANet_SettingContext.

11.7.7 Unsupported Attributes

SettingID is not used.

11.7.8 Methods

There are no supported methods for this class. To make changes to a setting, modify the required property and call PutInstance.

11.8 IANet_SettingString

11.8.1 Purpose

This class models a setting whereby the user can enter a free-form string value. For IANet_SettingMultiSelection, it is expected that the GUI will display an edit box.

11.8.2 Instances

An instance of this class exists for each setting that will be displayed as an edit box.

11.8.3 Creating instances

The user cannot create instances of this class.

11.8.4 Removing instances

The user cannot remove instances of this class.

11.8.5 Modifying properties

The “CurrentValue” attribute is the only modifiable property of this class. Modify this property by using Put() to change the value, then call “PutInstance()” to update the setting.

11.8.6 Associations

Each IANet_SettingMultiSelection instance is associated with an IANet_ElementConfiguration instance using an instance of IANet_SettingString.

11.8.7 Methods

There are no supported methods for this class.

11.8.8 Unsupported Attributes

SettingID is not used.

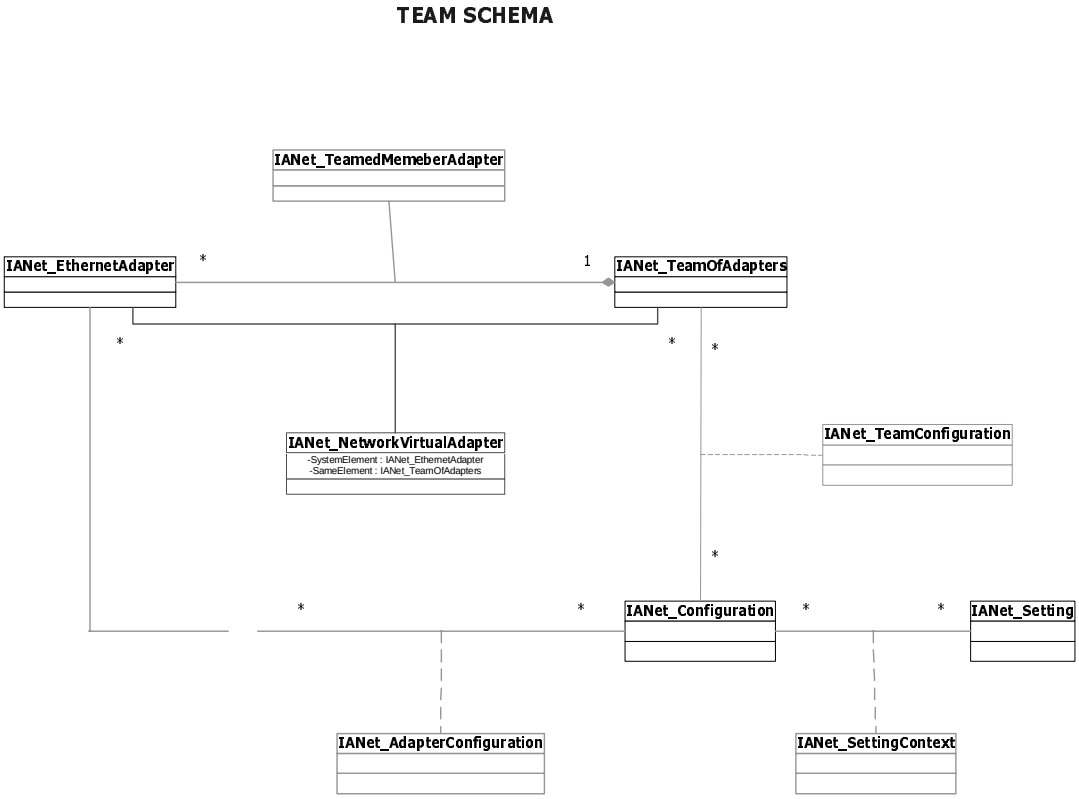
11.8.9 Methods

There are no supported methods for this class. To make changes to a setting, modify the required property, then call PutInstance.

12. Team Schema

The Team Schema describes how the Ethernet adapters are grouped together into teams.

12.1 Schema Diagram



12.2 IANet_TeamOfAdapters

12.2.1 Purpose

This class implements the CIM_RedundancyGroup class. This class has members that describe the type of the team, the number of adapters in the team, and the maximum number of adapters that can be in the team.

12.2.2 Instances

There is an instance of this class for each Intel Team.

12.2.3 Creating instances

To create an empty team, the user will create an instance of IANet_TeamOfAdapters. The user must set the correct “TeamingMode” before calling IWbemServices::PutInstance() to create the object in the Provider. The Provider will return a string containing the object path of the new object.

12.2.4 Removing instances

Correspondingly, to remove a team the user should delete the instance of IANet_TeamOfAdapters. The Provider will delete the associations to the team members, and will also delete the virtual adapter and settings for the team.

12.2.5 Modifying properties

Use Put() to change the value of the "TeamingMode" property, then call PutInstance() to update the team.

12.2.6 Associations

Each adapter in a team is associated with the team's instance of IANet_TeamOfAdapters using an instance of IANet_TeamMemberAdapter.

The virtual adapter for the team is associated with this class using an instance of IA_NetNetworkVirtualAdapter.

12.2.7 Methods

This class instance supports following methods:

TestSwitchConfiguration	Tests the switch configuration to ensure that the team is functioning correctly with the switch. This test can be used to check that link partners i.e., a device that an adapter links to, such as another adapter, hub, switch, etc., support the chosen adapter teaming mode. For example, if the adapter is a member of a Link Aggregration team, then this test can verify that link partners connected to the adapter support Link Aggregation.

12.3 IANet_TeamedMemberAdapter

12.3.1 Purpose

This class is used to associate the adapter with the team, and determines the function of the adapter in the team, and establishes that the adapter is currently active in the team. This class implements the CIM class CIM_NetworkAdapterRedundancyComponent.

12.3.2 Instances

An instance of this class exists for each adapter that is a member of a team.

12.3.3 Creating instances

To add an adapter to a team, create an instance of IANet_TeamedMemberAdapter to associate the adapter with the team.

12.3.4 Removing instances

To remove an adapter from the team, remove the instance of IANet_ TeamedMemberAdapter. The adapter will no longer be part of the team and may be bound to an IP protocol endpoint after the Apply() function is called.

12.3.5 Modifying properties

The "AdapterFunction" property of this class may be modified to describe how the adapter is used within a team.

12.3.6 Associations

This is an association class.

12.3.7 Methods

There are no supported methods on this class.

12.4 IANet_NetworkVirtualAdapter

12.4.1 Purpose

This class is used to associate the team's IANet_TeamOfAdapters with the IANet_EthernetAdapter that represents the virtual adapter for the team. The class implements the CIM class CIM_ CIM_NetworkVirtualAdapter.

12.4.2 Instances

An instance of this class exists for each Intel team that has been bound to a virtual adapter.

12.4.3 Creating instances

The user cannot create instances of this class. To create a team the user creates an instance of IANet_TeamOfAdapters. This class will not exist until after the user has called IANet_NetService .Apply() within the context of a valid session and the IANet_EthernetAdapter instance has been created.

12.4.4 Removing instances

The cannot delete instances of this class.

12.4.5 Associations

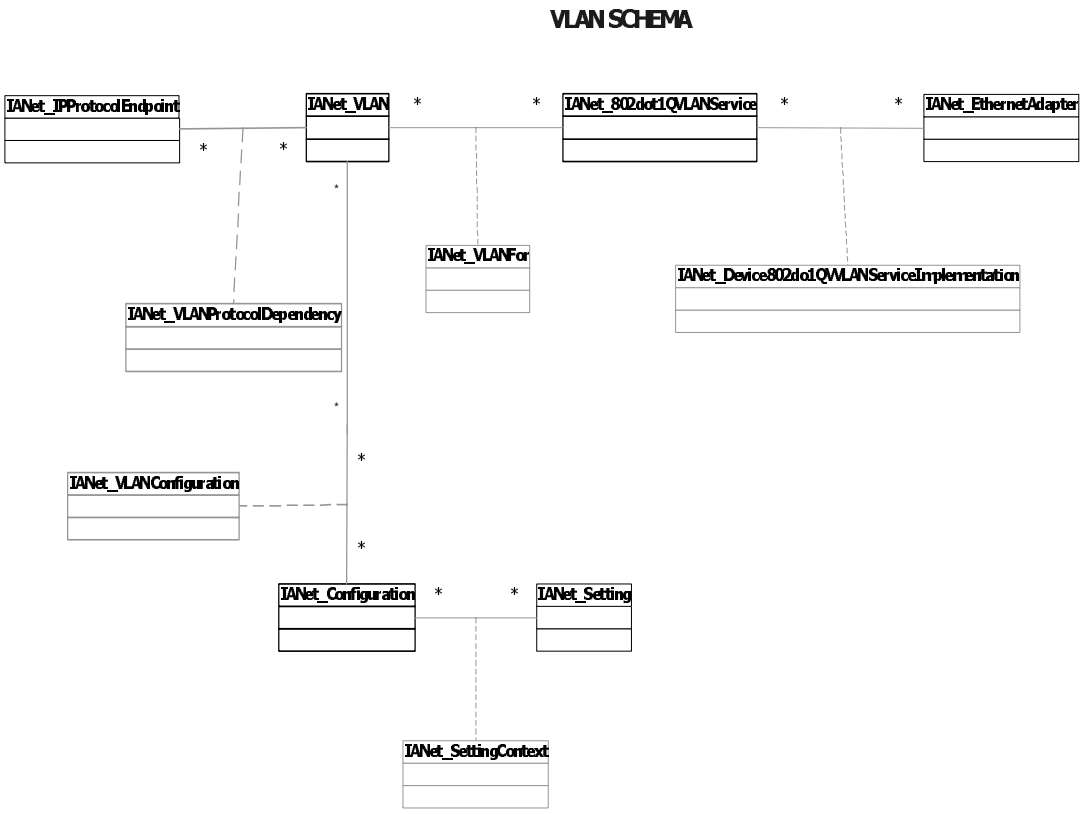
This is an association class.

12.4.6 Methods

There are no supported methods on this class.

13. VLAN Schema

13.1 Schema Diagram



13.2 IANet_802dot1QVLANService

13.2.1 Purpose

This class is used to hold the IEEE 802.1Q properties of a network adapter. The class implements the CIM class CIM_802dot1QVLANService.

13.2.2 Instances

An instance of this class exists for each adapter or team that supports IEEE 802.1Q. Each adapter can have just one IANet_802dot1QVLANService. Some teams, such as multi-vendor fault tolerant teams do not support this service.

13.2.2.1 Anomaly

A team with no VLANs does not have a VLAN service unless the user enumerates it within the context of a valid session. For teams, the 802.3QvlanService instance will appear only in following situations:

- if the team already has VLANs
- if the team does not have VLANs and user uses a session handle in the context while enumerating this class.

The above situations exist exclusively because if a team has no VLANs the Provider must do a pre-check to create a VLAN to see if the team supports them. This pre-check always fails if there is no session handle.

13.2.3 Creating instances

The user cannot create instances of this class If the adapter does not have an instance associated with it, then the adapter does not support this service.

13.2.4 Removing Instances

The user cannot delete instances of this class.

13.2.5 Modifying properties

There are no modifiable properties of this class.

13.2.6 Associations

Each instance of this class will be associated with one IANet_EthernetAdapter using an instance of IANet_DeviceServiceImplementation.

Each instance of IANet_802dot1QVLANService can support several VLANs; each VLAN will be associated with the instance using IANet_VLANFor association.

13.2.7 Methods

uint16 CreateVLAN([in] uint32 VLANNumber, [in] string Name, [out] IANet_VLAN REF VLANpath);	Used to create a VLAN on the adapter or team. The client must supply the VLAN number and the VLAN name, and will get the object path of the newly created VLAN.

13.3 IANet_VLAN

13.3.1 Purpose

This class holds the information for each Intel VLAN. This class implements CIM_VLAN.

13.3.2 Instances

An instance of this class will exist of each Intel VLAN.

13.3.3 Creating instances

To create a VLAN, call CreateVLAN on the appropriate instance of IANet_802dot1QVLANService.

13.3.4 Removing Instances

The user can remove an instance of this class to remove the corresponding VLAN.

13.3.5 Modifying properties

The user is able to modify the VLANNumber and Caption attribute.

13.3.6 Associations

Each instance is associated with an instance of IANet_802dot1QVLANService (and therefore to one instance of IANet_EthernetAdapter) using the class IANet_VLANFor.

Each instance can be associated with several IANet_Configuration instances to group together a set of settings for the VLAN. For this Provider release, there is only one IANet_Configuration object per VLAN.

Each instance can be associated with one IANet_IPProtocolEndpoint to provide the IP settings for the VLAN using the class IANet_VLANProtocolDependency.

13.3.7 Methods

None

14. Getting the current configuration

The client does not need to get a session handle to read the current configuration. Clients can use a NULL context, however, any error messages will be returned in the default language for the managed machine.

In the following table, items enclosed in { } are object paths. These paths are assumed to have been obtained from previous WQL queries. The client should never need to construct an object path without doing a query. The **__PATH** attribute of every object contains the object path for that object.

In all the following use cases, the methods IWbemServices::ExecQuery or IWbemServices::ExecQueryAsync are used to execute WQL queries.

14.1 Getting the Physical Adapters

The main class for the adapters is IANet_EthernetAdapter. This class is used for both physical and virtual adapters, and the client needs to know how to distinguish between them.

Task	WQL Query	Result Class	Comment
Enumerate all adapters	SELECT * FROM IANet_EthernetAdapter	IANet_EthernetAdapter	Returns all IANet_EthernetAdapters. This is equivalent to IWbemServices::CreateInstanceEnumAsync.
Determine if adapter is virtual	ASSOCIATORS OF {adapter path} WHERE AssocClass = IANet_NetworkVirtualAdapter	IANet_TeamOfAdapters	If the query results in no classes then the adapter is a real adapter.
Determine if adapter is a phantom adapter	ASSOCIATORS OF {adapter path} WHERE ResultClass = IANet_EthernetPCIDevice	IANet_EthernetPCIDevice	If the adapter is not virtual and this query returns no objects then the adapter is a phantom adapter.

14.2 Getting the PCI Devices

The main classes here are IANet_EthernetPCIDevice, IANet_PCIDevice, and IANet_AdapterDevice (an association class to associate an adapter with its device).

In this case, the association class does not contain any data, i.e., it has no value by itself. IANet_EthernetPCIDevice inherits from IANet_PCIDevice and contains extra attributes that are specific for a PCI device that is an Ethernet adapter.

Task	WQL Query	Result Class	Comment
Enumerate network PCI devices	SELECT * FROM IANet_PCIDevice	IANet_PCIDevice	The query may return modems and other devices that Intel® PROSet does not configure. This is equivalent to IWbemServices::CreateInstanceEnumAsync.
Enumerate all PCI devices used by Intel® PROSet configured adapters.	SELECT * FROM IANet_EthernetPCIDevice	IANet_EthernetPCIDevice	The query returns just the PCI devices that Intel® PROSet manages. This is equivalent to IWbemServices::CreateInstanceEnumAsync.

Task	WQL Query	Result Class	Comment
Determine if adapter is installed.	N/A	IANet_EthernetPCIDevice	Check the "Availability" attribute of IANet_EthernetPCIDevice. If equal to 10 - "Not Installed", then the device is not installed. Note: Not much information is known about the device at this stage.
Get the PCI Device associated with an adapter	ASSOCIATORS OF { IANet_EthernetAdapter path} WHERE ResultClass = IANet_EthernetPCIDevice	IANet_EthernetPCIDevice	If this returns no objects, then the adapter is either virtual or a phantom adapter.
Get the adapter associated with a PCI device	ASSOCIATORS OF {Ethernet PCI device path} WHERE ResultClass = IANet_EthernetAdapter	IANet_EthernetAdapter	This query is not optimal for the Provider – it is preferred that the client starts with the adapter.

14.3 Getting the Adapter Settings

The setting objects are not associated with the adapter directly. In conformance with the CIM standard, they are associated with a configuration object, which is associated with the adapter.

The classes involved with this part of the schema are IANet_EthernetAdapter, IANet_Configuration, IANet_SettingInt, IANet_SettingString, IANet_SettingEnum, IANet_SettingMultiSelection and IANetSettingSlider.

The association classes IANet_AdapterConfiguration and IANet_SettingContext do not contain any real data – they act as glue between the settings and their parent object.

Task	WQL Queries	Result Class	Comments
Get the Configuration object for the adapter	ASSOCIATORS OF {IANet_EthernetAdapter path} WHERE ResultClass = IANet_Configuration	IANet_Configuration	Returns exactly one object; even if there are no settings there will always be a configuration object. The object path from this object is used in the next query.
Get the settings associated with an adapter	ASSOCIATORS OF {IANet_Configuration path} WHERE AssocClass = IANet_SettingContext	A mixture of IANet_SettingInt, IANet_SettingString, IANet_SettingEnum, IANet_SettingMultiSelection, IANet_SettingSlider	Returns all the setting classes associated with the adapter. The client should use the __CLASS attribute to determine that type of each setting.

14.4 Getting the team configuration

The main classes in the teaming schema are IANet_EthernetAdapter, IANet_TeamOfAdapters, IANet_NetworkVirtualAdapter and IANet_TeamedMemberAdapter. The challenge in this schema is that an instance of IANet_EthernetAdapter exists for each physical adapter and for each virtual adapter. The client must be capable of distinguishing between the virtual adapter for a team and the adapters that are members of the team.

The association class IANet_NetworkVirtualAdapter contains no useful data – clients are really only interested in the endpoints of this association. IANet_TeamedMemberAdapter does contain useful data about how the member adapter is used within the team.

Task	WQL Queries	Result Class	Comments
Enumerate all teams	SELECT * FROM IANet_TeamOfAdapters	IANet_TeamOfAdapters	There is one instance of IANet_TeamOfAdapters for each team. This is equivalent to IWbemServices::CreateInstanceEnumAsync.
Get the virtual adapter for a team	ASSOCIATORS OF {IANet_TeamOfAdapters path} WHERE AssocClass = IANet_NetworkVirtualAdapter	IANet_EthernetAdapter	Returns only the adapter object for the virtual adapter in the team. This adapter will not exist if the team has been created but Apply has not been called. (see below on updating the configuration).
Enumerate the team's member adapters	ASSOCIATORS OF {IANet_TeamOfAdapters path} WHERE AssocClass = IANet_TeamedMemberAdapter	IANet_EthernetAdapter	Returns the adapters which are in the team, but does not describe what role the adapter plays.
Determine an adapter's role in a team	REFERENCES OF {IANet_EthernetAdapter path} WHERE ResultClass = IANet_TeamedMemberAdapter	IANet_TeamedMemberAdapter	The class contains information about how the member adapter relates to the team and its current status within the team.

14.5 Getting the team settings

The setting objects are not associated with the team directly. In conformance with the CIM standard, they are associated with a configuration object which is associated with the virtual IANet_EthernetAdapter for the team. The same configuration object is also associated with the IANet_TeamOfAdapters object for the team.

The classes involved with this part of the schema are IANet_EthernetAdapter, IANet_TeamOfAdapters, IANet_Configuration, IANet_SettingInt, IANet_SettingString, IANet_SettingEnum, IANet_SettingMultiSelection and IANetSettingSlider.

The association classes IANet_AdapterConfiguration and IANet_SettingContext do not contain any real data – they act as glue between the settings and their parent object. This is exactly the same as the adapter settings case.

Task	WQL Queries	Result Class	Comments
Get the configuration object for the team, starting with the virtual adapter	ASSOCIATORS OF {IANet_EthernetAdapter path} WHERE ResultClass = IANet_Configuration	IANet_Configuration	Returns exactly one object. Even if there are no settings, there will always be a configuration object. The object path from this object is used in the next query.
Get the configuration object for the team, starting with the team of adapters	ASSOCIATORS OF {IANet_TeamOfAdapters path} WHERE ResultClass = IANet_Configuration		

Task	WQL Queries	Result Class	Comments
Get the settings associated with an adapter	ASSOCIATORS OF {IANet_Configuration path} WHERE AssocClass = IANet_SettingContext	A mixture of IANet_SettingInt, IANet_SettingString, IANet_SettingEnum, IANet_SettingMultiSelection, IANet_SettingSlider	Returns all the setting classes associated with the adapter. The client should use the __CLASS attribute to determine that type of each setting.

14.6 Getting the VLAN configuration

Each adapter that supports VLANs has an IANet_802dot1QVLANService associated with it, using the association class IANet_Device802do1QVVLANServiceImplementation. If an adapter does not have an instance of this class associated with it, then it does not support VLANs.

Each VLAN is represented by an instance of IANet_VLAN. The VLAN is not directly associated with the adapter – it is associated with the IANet_802dot1QVLANService for the adapter.

The association class IANet_VLANFor is used to associate each VLAN instance with the correct IANet_802dot1QVLANService. This class contains no useful data for the user.

Task	WQL Queries	Result Class	Comments
Get the 802.1q VLAN service object associated with an adapter	ASSOCIATORS OF {IANet_EthernetAdapter path} WHERE ResultClass = IANet_802dot1QVLANService	IANet_802dot1QVLANService	Returns one or no object(s).
Get the VLANs on an adapter	ASSOCIATORS OF {IANet_802dot1QVLANService path} WHERE ResultClass = IANet_VLAN	IANet_VLAN	This can return no objects if there are no VLANs installed.

14.7 Getting the VLAN settings

The setting objects are not associated with the VLAN directly. In conformance with the CIM standard, they are associated with a configuration object, which is associated with the IANet_VLAN object for the VLAN.

The classes involved with this part of the schema are IANet_VLAN, IANet_Configuration, IANet_SettingInt, IANet_SettingString, IANet_SettingEnum, IANet_SettingMultiSelection and IANetSettingSlider.

The association classes IANet_VLANConfiguration and IANet_SettingContext do not contain any real data – they act as glue between the settings and their parent object. This is exactly the same as the adapter settings case.

Task	WQL Queries	Result Class	Comments
Get the Configuration object for the VLAN	ASSOCIATORS OF {IANet_VLAN path} WHERE ResultClass = IANet_Configuration	IANet_Configuration	Returns exactly one object; even if there are no settings there will always be a configuration object. The object path from this object is used in the next query.
Get the settings associated with a VLAN	ASSOCIATORS OF {IANet_Configuration path} WHERE AssocClass = IANet_SettingContext	A mixture of IANet_SettingInt, IANet_SettingString, IANet_SettingEnum, IANet_SettingMultiSelection, IANet_SettingSlider	Returns all the setting classes associated with the VLAN. The client should use the __CLASS attribute to determine the type of each setting.

14.8 Getting the IP Protocol Information

The Provider will give some limited information about the IP Protocol Endpoints that are associated with adapters, VLANs and teams. No other protocols are supported.

The main class containing the protocol information is IANet_IPProtocolEndpoint. There are two association classes: IANet_VLANProtocolDependency and IANet_AdapterProtocolImplementation. To get the IP endpoint for a team, first get the virtual IANet_EthernetAdapter for the team., i.e., the IP Endpoint is associated with this instance.

Task	WQL Queries	Result Class	Comments
Get the IP Protocol Endpoints associated with an adapter	ASSOCIATORS OF {path of IANet_EthernetAdapter} WHERE ResultClass = IANet_IPProtocolEndpoint	IANet_IPProtocolEndpoint	Though some adapters may have more than one IP address, they will be associated with only one IP Protocol Endpoint instance.
Get the IP Protocol Endpoints associated with a VLAN.	ASSOCIATORS OF {path of IANet_VLAN} WHERE ResultClass = IANet_IPProtocolEndpoint	IANet_IPProtocolEndpoint	There will be one IP Protocol Endpoint associated with the VLAN.

14.9 Getting the Boot Agent Information

Each adapter that can support a boot agent in flash ROM will have an IANet_BootAgent instance associated with it using the IANet_DeviceBootServiceImplementation association class.

Task	WQL Queries	Result Class	Comments
Get the Boot Agent associated with an adapter	ASSOCIATORS OF {path of IANet_EthernetAdapter} WHERE ResultClass = IANet_BootAgent	IANet_BootAgent	The following read only attributes provide information on the boot ROM image for this adapter: InvalidImageSignature, Version, UpdateAvailable, FlashImageType

14.10 Getting the BootAgent Settings

The setting objects are not associated with the boot agent directly. In conformance with the CIM standard, they are associated with a configuration object which is associated with the boot agent.

The classes involved with this part of the schema are IANet_BootAgent, IANet_Configuration, IANet_SettingInt, IANet_SettingString, IANet_SettingEnum, IANet_SettingMultiSelection and IANetSettingSlider.

The association classes IANet_BootAgentConfiguration and IANet_SettingContext do not contain any real data – they act as glue between the settings and their parent object.

Task	WQL Queries	Result Class	Comments

Task	WQL Queries	Result Class	Comments
Get the Configuration object for the boot agent	ASSOCIATORS OF { IANet_BootAgent path} WHERE ResultClass = IANet_Configuration	IANet_Configuration	Returns exactly one object. Even if there are no settings, there will always be a configuration object. The object path from this object is used in the following query.
Get the settings associated with an boot agent	ASSOCIATORS OF {IANet_Configuration path} WHERE AssocClass = IANet_SettingContext	A mixture of IANet_SettingInt, IANet_SettingString, IANet_SettingEnum, IANet_SettingMultiSelection, IANet_SettingSlider	Returns all the setting classes associated with the adapter. The client should use the __CLASS attribute to determine that type of each setting.

15. Updating the configuration

In most cases, to update the configuration, the client application will need to get a session handle from the IANet_NetService class and store this handle in an IWbemContext context object. Changes to the configuration will only occur when the "Apply" method on the IANet_NetService is called. There are some exceptions to this requirement:

- Changes to the boot agent class will occur immediately as they are made, and do not require a session handle.
- Certain method calls (e.g. identify adapter) cause an operation to be carried out before Apply is called.

For some operations the user can use the PreCheck qualifier in the context to check to see if an operation is allowed. This is to allow a user interface to disable certain controls or menu items, if required.

15.1 Changing the adapter, team or VLAN settings

Requires Session Handle	YES
Can use PreCheck	NO
Requires Apply call before operation is executed	YES

To change an adapter, VLAN or Team setting, the client must first get the object path of the setting that it will change. This is best done by enumerating the settings on the object and storing the **__PATH** attribute of the setting (see above).

The easiest way for the client to update a setting, is to: (1) get an instance of the setting object from WMI, (2) modify the **CurrentValue** attribute (using IWbemClassObject::Put()), and (3) call IWbemServices::PutInstance() to pass the modified instance back to the Provider. PutInstance must be called with the flag WBEM_FLAG_UPDATE_ONLY.

The Provider will validate the CurrentValue and return WBEM_E_FAIL if the validation failed. The exact reason for the failure will be returned in the **Description** attribute of the IANet_ExtendedStatus object.

Setting specific descriptions include:

- The integer setting value was less than the minimum allowed
- The integer setting value was greater than the maximum allowed
- The integer setting value is not one of the allowable steps
- The length of the string setting is bigger than the maximum allowed
- The setting value is not one of the allowable values

The last description is returned whenever the current value for IANet_SettingEnum, IANet_SettingSlider or IANet_SettingMultiSelection is not one of the allowable values.

The only attribute for a setting that the client can change is **CurrentValue**. The Provider will ignore changes made to any of the other values.

There are no supported methods on the setting class. To make changes to a setting modify the **CurrentValue** property, then call PutInstance.

15.2 Creating a new (empty) team

Requires Session Handle	YES
Can use PreCheck	YES
Requires Apply call before operation is executed	YES

To create a new team, create an instance of IANet_TeamOfAdapters (i.e., use IWbemServices::GetObject() to get a class object for IANet_TeamOfAdapters, and then use IWbemServices::SpawnInstance() to create an instance of this object).

Then, use IWbemClassObject::Put to set the **TeamMode** attribute in the instance to be the desired team type (e.g., AFT). Finally, call IWbemServices::PutInstance() to create the team, passing the flag WBEM_FLAG_CREATE_ONLY.

The object path for the new team is stored in the IWbemCallResultObject that is passed back to the user when the call has completed. The method IWbemCallResult::GetResultString will get the new object path.

If this action fails, the client should check the IANet_ExtendedStatus to get the failure reasons.

The virtual IANet_EthernetAdapter and IANet_IPProtocolEndpoint classes for the team are not available until after the call to Apply. The settings for the team can be accessed using the IANet_Configuration object associated with the new IANet_TeamOfAdapters.

15.3 Adding an adapter to a team

Requires Session Handle	YES
Can use PreCheck	YES
Requires Apply call before operation is executed	YES

To add an adapter to a team create an instance of IANet_TeamedMemberAdapter (i.e., use IWbemServices::GetObject() to get a class object for IANet_TeamedMemberAdapter , and then use IWbemServices::SpawnInstance() to create an instance of this object).

The following attributes in the object must be set using IWbemClassObject::Put() :

- **GroupComponent** must be set to be the full object path of the IANet_TeamOfAdapters to which the adapter is to be added;
- **PartComponent** must be set to be the full object path of the IANet_EthernetAdapter that is to be added to the team.

The following attributes may optionally be set:

- can be used to set the priority for the adapter in the team.

Finally, call IWbemServices::PutInstance() to add the adapter to the team, passing the flag WBEM_FLAG_CREATE_ONLY.

If this action fails, check IANet_ExtendedStatus for the error code.

15.4 Removing an adapter from a team

Requires Session Handle	YES
Can use PreCheck	YES
Requires Apply call before operation is executed	YES

To remove an adapter from a team, delete the IANet_TeamedMemberAdapter instance that associates the adapter to the team using IWbemServices::DeleteInstance()

If this action fails, check IANet_ExtendedStatus for the error code.

15.5 Deleting a team

Requires Session Handle	YES
Can use PreCheck	NO
Requires Apply call before operation is executed	YES

To delete a team, delete the IANet_TeamOfAdapters instance using IWbemServices::DeleteInstance()

If this action fails, check IANet_ExtendedStatus to get the error code.

15.6 Changing the mode of a team

Requires Session Handle	YES
Can use PreCheck	YES
Requires Apply call before operation is executed	YES

To change the mode of a team, get the instance of IANet_TeamOfAdapters for the team (e.g., use IWbemServices::GetObject using the object path of the team).

Then, use IWbemClassObject::Put to change the **TeamMode** attribute for the team. Finally, call IWbemClassObject:: PutInstance to tell the Provider to update the team mode, passing the flag WBEM_FLAG_UPDATE_ONLY.

If this action fails, check IANet_ExtendedStatus to get the error code.

15.7 Changing an adapter's priority within a team

Requires Session Handle	YES
Can use PreCheck	NO
Requires Apply call before operation is executed	YES

To change the priority of an adapter the client should first get the instance of IANet_TeamedMemberAdapter for the adapter. (e.g. use IWbemServices::GetObject using the object path).

The client can then use IWbemClassObject::Put to change the **AdapterFunction** attribute for the adapter. Finally the client needs to call IWbemClassObject:: PutInstance to tell the Provider to update adapter's priority.

If this action fails the client should check the IANet_ExtendedStatus for the error code.

15.8 Uninstalling an adapter

Requires Session Handle	YES
Can use PreCheck	YES
Requires Apply call before operation is executed	YES

To uninstall an adapter, call IWbemServices::DeleteInstance passing the object path of the adapter to uninstall.

15.9 Creating a VLAN

Requires Session Handle	YES
Can use PreCheck	YES
Requires Apply call before operation is executed	YES

To create a VLAN, call the CreateVLAN method on the IANet_802dot1QVLANService for the adapter to which the VLAN is to be added. The following arguments must be passed to the method:

- **VLANNumber** the number of the VLAN. (Range 1- 4094)
- **Name** a user definable name to identify the VLAN.

The function will return the object path of the newly created VLAN in the out parameter **VLANpath.**

If this action fails, check IANet_ExtendedStatus for the error code.

15.10 Changing the attributes of a VLAN

Requires Session Handle	YES
Can use PreCheck	YES
Requires Apply call before operation is executed	YES

The client can change the **VLANNumber** and **VLANName** attributes for a VLAN. To change the priority of an adapter, first get the instance of IANet_VLAN for the adapter (e.g. use IWbemServices::GetObject using the object path).

Then, change **VLANNumber** or **VLANName** to the desired values. . Finally, call IWbemClassObject:: PutInstance to tell the Provider to update the attributes, passing the flag WBEM_FLAG_UPDATE_ONLY.

If this action fails, check the IANet_ExtendedStatus for the error code.

15.11 Deleting a VLAN

Requires Session Handle	YES
Can use PreCheck	NO

Requires Apply call before operation is executed	YES

To delete a VLAN, call IWbemServices::DeleteInstance passing the object path of the VLAN to delete.

15.12 Updating the Boot Agent

Requires Session Handle	NO
Can use PreCheck	NO
Requires Apply call before operation is executed	NO

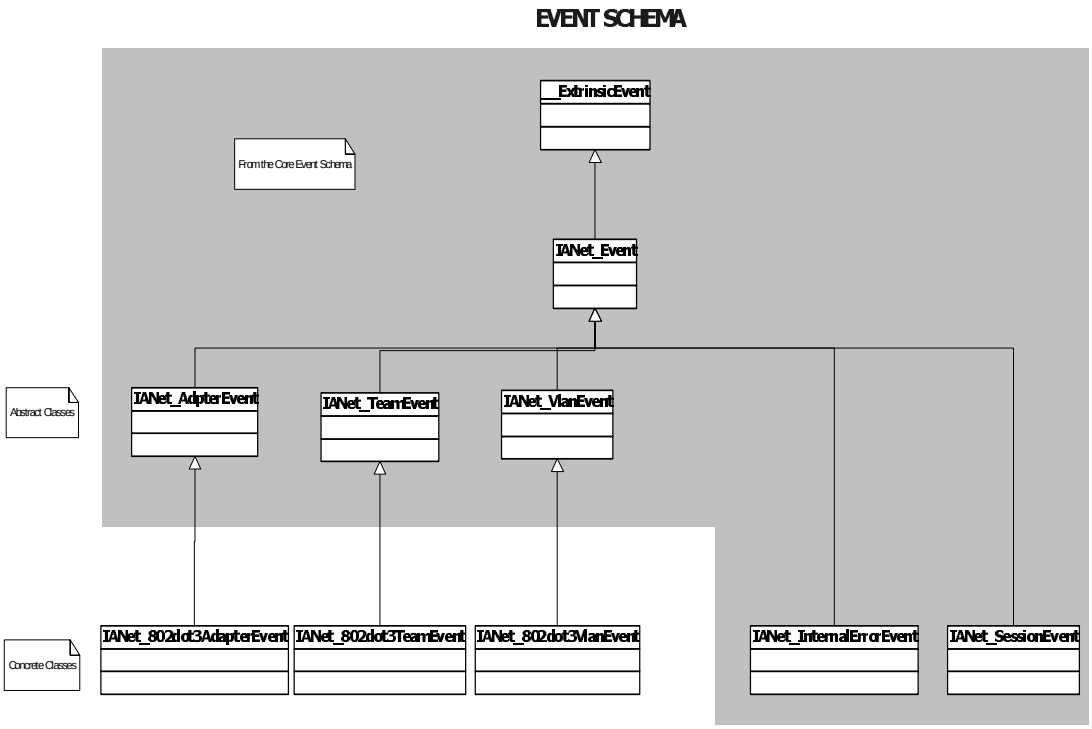
The client can update the Boot Agent Image by using methods calls. To read/write flash image, first get the instance of IANet_BootAgent for the adapter (e.g., use IWbemServices::GetObject using the object path).

Then, execute ReadFlash() to read the existing flash boot ROM image or ProgramFlash() to update the flash boot ROM image.

If this action fails, check the IANet_ExtendedStatus for the error code.

Task	WMI methods	Result	Comments
Update or Insert a boot ROM image for the adapter	uint32 ProgramFlash([IN, ValueMap {"0","1"} , Values {"Check Version", "Write Flash"}: Amended] uint32 Action, [IN] uint8 NewFlashData[], [OUT] string strErrorMessage);	If “Check Version” action is specified, this method will return with a warning message, if boot ROM image being updated as in NewFlashData[] is older than one already present on NIC. If “Write” action is specified, this updates the FLASH ROM on the NIC with NewFlashData[].	This method is used to update the Flash ROM on the NIC. This will cause the NIC to stop communicating with the network while the flash is updated.
Read boot ROM image	uint32 ReadFlash([OUT] uint8 FlashData[]);	FlashData[] contains the Flash ROM image on the NIC	This method reads the Flash ROM on the NIC which can be saved into a file.

16.1 Event Schema



16.2 Concrete Event Classes

16.3 IANet_802dot3AdapterEvent

16.3.1 Purpose

This event notifies the client about a change in the status or configuration of an adapter.

16.3.2 Triggers

The event occurs after an adapter status changes, or after a user has changed an adapter setting and called Apply.

16.3.3 Event Data

AdapterPath contains the object path of the adapter that caused the event.

16.4 IANet_802dot3TeamEvent

16.4.1 Purpose

This event notifies the client about a change in the status or configuration of a team.

16.4.2 Triggers

The event occurs:

- after an adapter status changes, or;
- after user has changed an team setting and called Apply, or;
- or after the team configuration has been changed and the user calls Apply.

16.4.3 Event Data

TeamPath contains the object path of the team that caused the event.

16.5 IANet_802dot3VlanEvent

16.5.1 Purpose

This event notifies the client about a change in the status or configuration of a VLAN.

16.5.2 Triggers

The event occurs:

- after a VLAN status changes, or;
- after user has changed a VLAN setting and called Apply, or;
- or after the VLAN configuration has been changed and the user calls Apply.

16.5.3 Event Data

VlanPath contains the object path of the VLAN that caused the event.

16.6 Registering For Events

Applications should use IWbemServices:: ExecNotificationQuery or IWbemServices:: ExecNotificationQueryAsync to request notifications for events. The following queries are examples of event notification queries. This list is not exhaustive as many queries are possible.

Task	Notification Query
Request all events	SELECT * FROM IANet_Event
Request all adapter events	SELECT * FROM IANet_AdapterEvent
Request all team events	SELECT * FROM IANet_TeamEvent
Request all session events	SELECT * FROM IANet_SessionEvent
Request all VLAN events	SELECT * FROM IANet_VlanEvent
Request all internal events	SELECT * FROM IANet_InternalErrorEvent
Request adapter events for a particular adapter	SELECT * FROM IANet_AdapterEvent WHERE AdapterPath={IANet_EthernetAdapter object path}

17. Optimized WQL Queries

17.1 Settings

The Provider is optimized to allow applications to get settings using queries. The Provider can recognize the following queries, and will only return the objects that match. All other queries will result in the Provider getting all the settings on all objects, and CIMOM will filter them before they reach the application. With several adapters, teams and VLANs this can cause a delay of several seconds when retrieving required data.

17.1.1 Getting the settings for a particular adapter, VLAN or Team

The following query will get only the settings for a particular adapter, VLAN or Team. WQL does not allow any additional clauses in the WHERE clause.

ASSOCIATORS OF {IANet_Configuration path} WHERE AssocClass = IANet_SettingContext

17.1.2 Getting one setting

The following query can be used to get a single setting for an object without querying to get them all:

SELECT * FROM <SETTING CLASS> WHERE ParentId="<Device ID>" AND ParentType="<type>" AND Caption="<SETTING NAME>"

Notes:

- The class must be the exact setting class – not a base class (e.g. IANet_SettingInt).
- Acceptable ParentTypes are "NIC", "Team" , "VLAN" or “BootAgent”.
- ParentId is the GUID that uniquely defines the object with the setting (for an adapter, this is the DeviceId).
- This is not a recommended method of getting the settings associated for an object; the preferred method is use associations. However, WQL does not support the complex query required (i.e., WQL does not support ASSOCIATORS OF {IANet_Configuration path} WHERE AssocClass = IANet_SettingContext AND Caption="<SETTING NAME>")

18. Summary of CIM classes

Class	Can Create?	Can Delete?	Implemented Methods	Settable Properties	Unsupported Properties	Instance Count	Related Association Classes
IANet_802dot1QVLANService	N	N	CreateVLAN	GRVPEnabled, JoinTime	Description, Install Date, Started, StartMode, Status	One instance for each team or adapter that supports VLANs	IANet_Device802dot1QVLANServiceImplementation, IANet_VLANFor
IANet_AdapterConfiguration	N	N	None	None	None	One instance for each adapter.	This class associates IANet_EthernetAdapter with IANet_Configuration.
IANet_AdapterDevice	N	N	None	None	None	One instance for each non-phantom adapter	This class associates IANet_EthernetAdapter with IANet_EthernetPCIDevice
IANet_AdapterProtocolImplementation	N	N	None	None	None	One instance for each IP protocol endpoint which is bound to an adapter.	This class associates IANet_EthernetAdapter with IANet_IPProtocolEndpoint
IANet_BootAgent	N	N	ProgramFlash ReadFlash	None	Caption, Description, InstallDate, Started, StartMode, Status	One instance for each adapter that supports the boot agent capability.	IANet_DeviceBootServiceImplementation, IANet_BootAgentConfiguration
IANet_BootAgentConfiguration	N	N	None	None	None	One instance for each boot agent.	This class associates IANet_BootAgent with IANet_Configuration.
IANet_Configuration	N	N	None	None	None	One Instance for each adapter, VLAN and team.	IANet_AdapterConfiguration, IANet_VLANConfiguration, IANet_SettingContext

Class	Can Create?	Can Delete?	Implemented Methods	Settable Properties	Unsupported Properties	Instance Count	Related Association Classes
IANet_Device802dot1QVLANServiceImplementation	N	N	None	None	None	One instance for each adapter or team which supports VLANs	This class associates IANet_EthernetAdapter with IANet_802dot1QVLANService.
IANet_EthernetAdapter	N	Y	IdentifyAdapter HasVLANs IsPowerMgmtSupported GetPowerUsage SetPowerUsage GetPowerUsageOptions SetPowerUsageOptions TestCable AdvancedTestCable TestLinkSpeed	None	AutoSense – (this is exposed as a setting), ErrorCleared, OtherIdentifyingInfo, IdentifyingDescriptions, InstallDate, LastErrorCode, MaxDataSize, MaxQuiesceTime, PowerManagementCapabilities– (this is exposed as a method), PowerManagementSupported– (this is exposed as a method), PowerOnHours , ShortFramesReceived, SymbolErrors, TotalPowerOnHours	One for each Intel®PROSet supported installed adapter, phantom adapter and team.	IANet_AdapterProtocolImplementation, IANet_AdapterDevice, IANet_AdapterConfiguration, IANet_TeamedMemberAdapter, IANet_NetworkVirtualAdapter, IANet_Device802dot1QVLANServiceImplementation, IANet_DeviceBootServiceImplementation
IANet_EthernetPCIDevice	N	N	None	None	AdditionalAvailabitlity, Capabilities,,CapabilityDescriptions, Caption, Description, DeviceSelectTiming, ErrorCleared, ErrorDescription,	One instance for each PCI card that is a Intel® PROSet supported Ethernet adapter.	IANet_AdapterDevice

Class	Can Create?	Can Delete?	Implemented Methods	Settable Properties	Unsupported Properties	Instance Count	Related Association Classes
					IdentifyingDescription, InstallDate, LastErrorCode, MaxNumberController, MaxQuiesceTime, Name, OtherIdentifyingInfo, PowerManagementCapabilities, PowerManagementSupported, PowerOnHours, ProtocolDescription, ProtocolSupported, SelfTestEnabled, Status, StatusInfo, TimeOfLastReset, TotalPowerOnHours		
IANet_IPProtocolEndpoint	N	N	None	None	Caption, Description, InstallDate, NameFormat, OtherTypeInformation, Status	One instance for each binding of the IP Protocol stack to an Intel supported endpoint.	IANet_AdapterProtocolImplementation, IANet_VLANProtocolDependency
IANet_NetService	N	N	GetSessionHandle, Apply, ReleaseSessionHandle, Cancel	None	Caption, Description, Install Date, Started, Start Mode, Status	Exactly one.	None

Class	Can Create?	Can Delete?	Implemented Methods	Settable Properties	Unsupported Properties	Instance Count	Related Association Classes
IANet_NetworkVirtualAdapter	N	N	None	None	None	One instance for each team.	This class associates IANet_TeamOfAdapters with an IANet_EthernetAdapter.
IANet_PCIDevice	N	N	None	None	AdditionalAvailability, Capabilities,,CapabilityDescriptions, Caption, DeviceSelectTiming, ErrorCleared, ErrorDescription, IdentifyingDescription, InstallDate, LastErrorCode, MaxNumberController, MaxQuiesceTime, Name, OtherIdentifyingInfo, PowerManagement Capabilities, PowerManagement Supported, PowerOnHours, ProtocolDescription, ProtocolSupported, SelfTestEnabled, TimeOfLastReset, TotalPowerOnHours	One instance for each PCI card that is a network device in the system. (For IA64, only the PCI devices which are Intel® PROSet supported adapters will have instances).	
IANet_Setting	N	N	None	None	SettingID	This is an abstract class.	IANet_SettingContext
IANet_SettingContext	N	N	None	None	None	One instance for each setting	This class associates IANet_Setting with an IANet_Configuration
IANet_SettingInt	N	N	None	CurrentValue	SettingID	One instance for each integer	IANet_SettingContext

Class	Can Create?	Can Delete?	Implemented Methods	Settable Properties	Unsupported Properties	Instance Count	Related Association Classes
						setting	
IANet_SettingMultiSelection	N	N	None	CurrentValue	SettngID	One instance for each multi-selection setting	IANet_SettingContext
IANet_SettingSlider	N	N	None	CurrentValue	SettingID	One instance for each slider setting	IANet_SettingContext
IANet_SettingString	N	N	None	CurrentValue	SettingID	One instance for each string setting	IANet_SettingContext
IANet_TeamedMemberAdapter	Y	Y	None	AdapterFunction	PrimaryAdapter, ScopeOfBalancing	One instance for every adapter which is in a team	This class associates IANet_TeamOfAdapters with an IANet_EthernetAdapter.
IANet_TeamOfAdapters	Y	Y	TestSwitchConfiguration	TeamingMode	Install Date, Status	One instance for each team	IANet_NetworkVirtualAdapter, IANet_TeamedMemberAdapter
IANet_VLAN	N	Y	None	VLANNumber, Caption	Description, Install Date, StartMode, Status	One instance for each VLAN	IANet_VLANFor
IANet_VLANConfiguration	N	N	None	None	None	One instance for each VLAN	This class associates IANet_VLAN with IANet_Configuration
IANet_VLANFor	N	N	None	None	None	One instance for each VLAN	This class associates IANet_VLAN with IANet_802dot1QVLANService.
IANet_VLANProtocolDependency	N	N	None	None	None	One instance for each VLAN.	This class associates IANet_VLAN with IANet_IPProtocolEndpoint